

WADDON
MODULAR RAILWAY CONTROL

By

David A. Graham

BSc (Hons) Computing

Bournemouth University

2007/2008

Project Supervisor: Dr Peter J. Knaggs

School of Design, Engineering and Computing

A b s t r a c t

This project seeks to automate a model railway layout. In doing this it discovers the benefits of control abstraction by building a microcontroller based control system that operates a small demonstration layout. The project concludes with a solution to automating a model railway layout.

D e c l a r a t i o n

This Project Report is submitted in partial fulfilment of the requirements for an honours degree at Bournemouth University. I declare that this Project Report is my own work and that it does not contravene any academic offence as specified in the University's regulations.

Retention

I agree that, should the University wish to retain it for reference purposes, a copy of my Project Report may be held by Bournemouth University normally for a period of 3 academic years. I understand that my Project Report may be destroyed once the retention period has expired. I am also aware that the University does not guarantee to retain this Project Report for any length of time (if at all) and that I have been advised to retain a copy for my future reference.

Confidentiality

I confirm that this Project Report does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or personal life has been anonymised unless permission for its publication has been granted from the person to whom it relates.

Copyright

The copyright for this Project Report remains with me.

Requests for Information

I agree that this Project Report may be made available as the result of a request for information under the Freedom of Information Act.

Signed:

Name: David Graham

Date: 28/03/08

Programme: BSc (Hons) Computing

A c k n o w l e d g m e n t s

I would firstly like to thank my brother Paul Graham for the introduction to microcontrollers and inspiring me to pursue a microcontroller based project.

I would also like to thank my brother Jonathan Graham for exposing me to computers and electronics (& Traffic Lights) from a young age. Thanks goes to all my family for the support through the recent hard times. Finally, I would like to thank my Supervisor Dr Peter Knaggs and Reader Chris Brown for their support and constructive feedback throughout the year.

G l o s s a r y

API: Application Programming Interface

BC: Block Controller

Block: A section of railway track.

CDU: Capacitor Discharge Unit

DIL: Dual In Line

IC: Integrated Circuit

IDE: Integrated Development Environment

mA: Milliampere

PCB: Printed Circuit Board

Point: A mechanical installation enabling trains to be guided from one track to another

PWM: Pulse-width Modulation

Signal: A signal is an electrical device erected beside a railway line to pass information relating to the state of the line ahead to train drivers

Track: Railway track

UART: Universal Asynchronous Receiver/Transmitter

USB: Universal Serial Bus

~ In Loving Memory of Mum ~

Table of Contents

Abstract	i
Declaration.....	ii
Acknowledgments.....	iii
Glossary	iv
Table of Contents	vi
1. Introduction.....	1
1.1. Background.....	1
1.2. Purpose.....	1
1.3. Objectives.....	1
2. Background Research.....	2
2.1. Introduction.....	2
2.2. The Principals of Control	2
2.2.1. Open-Loop Control	2
2.2.2. Closed-Loop Control	2
2.3. Control Technologies: The Microcontroller.....	3
2.3.1. Packaging and Appearance.....	3
2.3.2. Ports.....	3
2.3.3. Communications.....	4
2.3.4. Pulse-Width Modulation.....	4
2.4. Control Products: Lego Mindstorms	4
2.5. Types of Sensors	5
2.6. Abstraction.....	5
2.7. Routing for the Railways.....	7
2.8. Types of Computer Routing	7
2.8.1. Internet Protocol (IP).....	7
2.8.2. Asynchronous Transfer Mode (ATM).....	7
3. Method	9
3.1. Development Tools.....	9
3.2. Choice of Programming Language.....	9
3.3. Chosen Software Process Model.....	9
3.4. Design Approach.....	10
3.5. Developer Best Practice.....	10

4.	Requirements Analysis	11
4.1.	Requirements Gathering.....	11
4.2.	Elicitation from End Users	11
4.3.	Elicitation from Pre-existing Technologies.....	12
4.4.	Creation of Test Speciation	12
5.	Design Specification	13
5.1.	High Level Design.....	13
5.2.	The Demo Layout.....	13
5.3.	Train Movement	13
5.4.	Communications	14
5.5.	Managing Points.....	15
5.6.	Managing Signals.....	16
5.7.	Managing Opto Switches.....	16
5.8.	Selecting a Microcontroller.....	16
5.9.	Block Controller.....	17
5.10.	Software Design	17
5.10.1.	Overview	17
5.10.2.	APIs	18
5.10.3.	Multitasking	19
5.10.4.	State Machine	19
5.11.	Operator Control	19
6.	Implementation	21
6.1.	Overview	21
6.2.	Implementation Issues	21
6.2.1.	Power Supply Issue.....	21
6.2.2.	H-Bridge Temperature.....	21
6.2.3.	Darlington Driver Failure.....	21
6.2.4.	Low Voltage Programming	22
6.2.5.	Configuration Bits.....	22
6.2.6.	Master Clear.....	22
6.2.7.	PIC to PIC Communications.....	22
6.2.8.	Power Supply Issue II	22
6.2.9.	Memory Issues	22
6.2.10.	Memory Issues II	23

7.	Testing	24
7.1.	Unit Testing	24
7.2.	Integration Testing	24
7.3.	Black Box Testing.....	24
7.4.	Strategy	24
7.5.	Defects.....	25
7.6.	Post Mortem.....	25
8.	Evaluation	26
8.1.	Background Research Evaluation.....	26
8.2.	Design Evaluation.....	26
8.2.1.	Feedback	26
8.2.2.	Communications	26
8.2.3.	Movement Logic.....	27
8.2.4.	Development and Upgrades.....	27
8.3.	Implementation Evaluation.....	28
8.3.1.	Communications	28
8.3.2.	Possible Lockups	28
8.3.3.	Over Engineered.....	28
8.4.	Project Plan and Timeline.....	28
8.5.	Further Developments	29
8.5.1.	Expand I/O Capability.....	29
8.5.2.	Level Crossing Block Controller	30
8.5.3.	User Speed Control	30
8.5.4.	Implementation of USB.....	30
8.5.5.	Information Aids	30
8.6.	Evaluation of Professionalism	30
9.	Conclusion	31
	Bibliography	33
	Appendix A: Software Process Models	34
	Appendix B: Requirements Document.....	36
1.	Problem domain description.....	36
1.1.	Introduction.....	36
1.2.	Problem frame.....	36
1.3.	Trains	36

1.4.	Points	36
1.5.	Points Motor.....	37
1.6.	Signals	37
1.7.	Track	37
1.8.	Movements	37
1.9.	Layout	37
1.10.	Requirements.....	38
1.10.1.	Commercial Constraints	38
1.10.2.	Design Constraints	38
1.10.3.	Functional - Ordinary.....	38
1.10.4.	Performance	38
1.10.5.	Speed.....	38
1.10.6.	Capacity	38
1.10.7.	Reliability.....	39
1.10.8.	Usability.....	39
	Appendix C: Design Specification.....	40
1.	System Wide.....	40
1.1.	High Level Design Dataflow Diagram	40
1.2.	Railway Track Block Layout.....	40
1.3.	Communications Protocol.....	41
1.4.	Communications Backus Naur Form	41
2.	Block Controller.....	42
2.1.	High Level Design Dataflow Diagram	42
2.2.	PIC18F2550 Pin Configuration	42
2.3.	Schematic Diagram.....	43
2.4.	Movement Routines Flow Diagram.....	44
2.5.	Receive Communications Flow Diagram.....	45
2.6.	Send Communications Flow Diagram.....	46
2.7.	Movement Routines	46
	Appendix D: Test Results	48
	Appendix E: Original Project Proposal.....	56
	Appendix F: Original Project Plan	58
	Appendix G: Predicted Project Timeline	63
	Appendix H: Actual Project Timeline.....	64

Appendix I: Project Diary..... 65
Appendix J: CD Contents..... 69

1. Introduction

This chapter gives an overview of this project and highlights its purpose. The project objectives are identified which will guide the project to a conclusion.

1.1. Background

Detail is a goal for a model railway enthusiast. Lots of money can be spent on getting a railway layout looking scaled down and realistic. Most components to create the desired effect can be bought off the shelf. The majority of layouts however lack the most realistic effect of all, the appearance that the trains are being driven by individuals. Modellers still want a certain amount of control but it is unrealistic to have total control over the whole layout. Automation has always been possible but has normally relied on custom made circuitry that is costly to make and also is not adaptable if the layout grows.

With the rapid reduction of the cost of computer equipment, the associated components that make up a computer system have also reduced. Microcontrollers have all the components of a basic computer, processor, ram and flash memory. These can be programmed for a desired effect but more importantly they can now be purchased for a couple of pounds. Ten years ago the same technology would have cost hundreds of pounds but because of their low cost they are often used for control purposes. It is therefore now more possible to create a model railway controller using microcontrollers as the main control technology.

1.2. Purpose

The project is formed of two parts. Firstly background research is made to discover the commonly used technologies and theories for control purposes. The background research will aid the second part of the project which is the creation of a demonstration model railway layout and control architecture. The knowledge and understanding gained from the two parts will satisfy the objectives of the project and the result will be the development of a solution to automate a model railway layout.

1.3. Objectives

The project will discover the benefits if any of using a modular control architecture when controlling a model railway layout.

2. Background Research

2.1. Introduction

The research of this project will be based around controlling a model railway. Model railways have a large similarity with their full scale counterpart and it makes sense to also study how full scale railways are controlled. It is very likely that similar problems faced by the full scale railways will be encountered by this project. Likewise routing technologies have been extensively designed and documented in the computer industry so these will also be reviewed. Control Engineering is a large subject on its own but we have to balance up the project requirements with the amount of detail we need to go into.

The following areas will be researched:

Control Technology

Microcontrollers

British Railways

Routing and Networking

2.2. The Principals of Control

According to Bateson (2001): A control system is a group of components that maintains a desired result by manipulating the value of another variable in the system.

2.2.1. Open-Loop Control

An open-loop control system does not have any feedback to what the result of the control is affecting. The accuracy of control is reliant on calibration carried out either when the control system was built or at the start of the system's initialisation routine. With no feedback the control system cannot correct any undesired results. Firing a Gun is a good example of an open-loop control system. The control happens in the aiming of the gun but once the gun is fired the bullet is on its own and its course cannot be corrected once it leaves the barrel. The Main advantage of this type of control is it is less expensive to create as the control is very simple because it excludes any error correction.

2.2.2. Closed-Loop Control

Closed-loop control has the benefit of having feedback on what it is controlling. It can constantly monitor its intended result with the actual result and decide how it can correct any unintended results. An oven is a good example of a feedback control system where it

must maintain a constant temperature. It is constantly monitoring the temperature and deciding whether it should apply more heat.

2.3. Control Technologies: The Microcontroller

PIC (Peripheral Interface Controller) Microcontrollers are integrated circuits (IC) intended to be used for control purposes and due to their inexpensive nature and wide availability they are very popular with hobbyists. A PIC can come with EPROM, ROM or Flash ROM memory making them ideal for many uses from prototyping to creating a final production product. Many different variants of microcontroller are available each with added or reduced functionality which are useful for different purposes. Cheap hardware programmers are available which allow developers to program and reprogram the chips both easily and quickly.

2.3.1. Packaging and Appearance

The size of a PIC can vary widely depending on its use. It is not usually the size of the PIC itself that determines the overall size but the size, number and spacing of the pins. A chip with dual in line (DIL) packaging are normally connected to a printed circuit board (PCB) via a socket meaning it can be easily removed without the need of a soldering iron. DIL is favoured by the hobbyist due to its ease of application and removal to and from a circuit board. Industry however requires an even smaller footprint of a PIC so instead uses a Small-Outline Integrated Circuit (SOIC) packaging. SOIC uses surface mount technology meaning that the IC is soldered directly onto the PCB. This technique is normally only used on an industrial scale as special placement machines and ovens are required to mount and heat the IC and solder paste.

2.3.2. Ports

A port is a means of input or output at the same voltage as the supply to the V_{DD} of a chip. The number of ports available depends on the size of the chip. Ports are generally named A, B, C...ect. Most ports are 8 bits wide meaning there can be a mixture of input and output pins that total 8, these are normally mapped to a signal pin on the chip. Certain pins can be multiplexed with alternate functions but only one function can be active at a time meaning the loss of some functionality depending on the role of the pin.

2.3.3. Communications

Communications can be achieved via serial ports which allow the PIC to communicate with other serial enabled devices. There are numerous protocols available like USART, SPI, I2C and CAN. The USART is typically used for simple communications and can be easily interfaced with a personal computer via its standard serial port.

2.3.4. Pulse-Width Modulation

Pulse-Width Modulation is used in circumstances where the output voltage has to vary. Changing the speed of a DC motor or changing the brightness of a light are examples of situations where we need to vary the voltage. The output of a microcontroller can only be on or off as it is a digital device. PWM simulates a varied voltage by rapidly alternating between a high and low (on and off) output signal. The voltage effectively isn't changed but the alternation creates an averaged out signal. PWM can be achieved using software taking into account of the system clock however just like every commonly used feature, PWM has been included as a hardware feature of most microcontrollers. The PWM pins of the microcontrollers couldn't alone power a DC motor so a full bridge driver IC would typically be used to provide both forward and reverse operations of a motor.

2.4. Control Products: Lego Mindstorms

The Lego Mindstorms is an existing control product that is designed for the home user. It is a technical toy intended to control Lego models. The core control piece is a brick called a RCX. Inside is a Renesas H8/300 microcontroller. The user doesn't have to worry about the internal workings of the RCX as the brick appears to be an overly large Lego brick with the normal brick connectors and mountings. It is intended for the RCX to be directly mounted onto the Lego model created. Firmware is downloaded from a PC onto the RCX via an infrared interface. After the firmware is downloaded the RCX needs no further interactions with the PC and it is not intended for a user to control it directly. The RCX can control up to three devices like motors and also can have three sensors connected to it. Using its infrared interface it is also able to communicate with a second RCX brick enabling inter-brick cooperation or competition. The sensors available are extensive with touch, light, rotation and temperature all available.

The Mindstorms is interesting to study for this project. It is a product intended for home use and it also controls a similar application as defined in this project. Something that could be adopted by this project is the way the RCX is programmed. Creating a control

system that will control a number of different scenarios will not be possible without allowing the user to configure or program the control system in some way. This requirement can handicap a system if the user feels the setup requirement is too demanding for them. A compromise is often to limit the system functionality to make the setup process easier but this normally defeats the main purpose of the system of being very functional and adaptable. Line by line coding can be seen as a way of programming for skilled coders so wouldn't be recommended for the use by home users. Keeping with the theme of Lego, Mindstorms uses virtual onscreen Lego bricks that represent different aspects of logic which forms the complete program logic. Its Robotic Invention System which uses this method aids the user to development RCX firmware in a simple, visual way.

2.5. Types of Sensors

There are many types of sensors. Some of the most common are Push, Toggle, Slide and Rotary which can be also seen as traditional switches. Reed Switches are sensors that close when a magnet is passed near them. Tilt sensors use mercury, a liquid metal, to close the circuit depending on the angle of the sensor. Opto switches normally detect infrared light which can be shone away from it but any close object will bounce it back. This type of sensor is ideal when the object that is being monitored can't have anything placed onto it, like a magnet. Opto switches are often used for tracking the movement of objects or for example detecting if doors are closed. The mentioned sensors so far are digital sensors meaning they only open or close a circuit. These are easy to interface with system as they will act like a standard switch. Temperature and Movement sensors are both analogue devices and they both change the resistance of the power. These are more difficult to interface as the analogue signal has to be converted to a digital one for the system to understand.

2.6. Abstraction

Abstraction is a process where a system is designed as a number of smaller components, where each component's implementation isn't known when viewing the system as a whole.

The use of abstraction has always been used in traditional forms of engineering. The assembly of a motor vehicle, for example, is formed of many complex components that are individually designed and built to eventually form a complete vehicle. The level of abstraction is to understand what a component requires to function and what it expels.

For a vehicle part, there will always be a requirement for a form of mounting to the body of the vehicle. It will expel what ever the function of the part does; a water pump, for example, will expel flowing water.

We can see in other disciplines of engineering the process of breaking down a product into smaller components is a very natural one. By deciding on a level of abstraction we can take a complex problem and break it down into simpler smaller problems that are easier to understand.

The process of designing a system with abstraction follows a path of high level abstraction where there is little detail of each component's implementation, to lower levels of abstraction where more detail is known. Using this practice means the development process isn't slowed down with fine detail when this level of detail isn't yet known to the design team. For example, the body of a car could be designed and built without knowing the colour of the dashboard.

The concept of breaking a bigger problem down into smaller sub problems isn't a new design strategy for Software Engineers. When developers want to create a new application they don't worry about low level hardware control because the Operating System already handles this. Developers have access to Application Programming Interfaces (API) which reduces complexity of development when using any low lying hardware. Creating layers does create a layer of 'bloat' however the speed of current computers and the benefits of rapid development over compensates for this minor disadvantage.

In software engineering we describe a component of software as a module and we can measure the module coupling which is to the extent modules are dependent upon each other. We have to consider to what extent a module would have to know about the implementation of another module to function correctly. The highest level coupling is when there is a lot of linking between modules and the developer would have to consider what implications to a module would arise if a change was made to another. High coupling we try to avoid as it goes against the modular principal where we should only have to change or develop a module at a time. We try to develop modules with high Cohesion however. Cohesion is to the extent the function that the module performs is unified, integrated and well-defined. If the module carries out a well defined function it means we can easily remove, replace or repair the module without the fear of breaking another. This forms the major benefits of abstraction that parallel development and general maintenance possible. Other benefits of abstraction and modularisation includes

the ability to easily unit test the individual modules which eradicate whole sections where defects could occur. The ability that a module can be reused reduces design, coding and testing of a system which is a huge benefit especially when time is short.

2.7. Routing for the Railways

Railway routing hasn't really changed that much over the years. Britain has the oldest railway network in the world and as such most of the world's railways are based on the British approach.

Safety is a priority on the railways and keeping trains apart from each other lessens the likelihood of any accidents. Making sure trains only travel in a certain direction on a railway line improves safety even more but this is only possible when there is more than one line. Block signalling is technique to separate trains on the same line. The principle is fairly simple; a railway line is divided into sections known as 'Blocks'. Only one train can occupy a Block at a time. A block is controlled by only one signal box and any train sent into a block has to get permission from the controlling signal box.

A major disadvantage of this approach is that the block is always a fixed size and will always be longer than the longest train length. If a mix of train lengths uses the line then the shorter trains would waste the capacity of the block. A solution to the problem of fixed blocks is to use Moving Blocks. These require a more intelligent computer system to calculate a train's block size depending on the length of the train and its stopping distance.

2.8. Types of Computer Routing

In this section we only describe the basic principals of computer networking to see if we can extract any useful theories that can be translated onto a control domain.

2.8.1. Internet Protocol (IP)

The IP protocol transmits packets from sender to receiver often via intermediate routers. The route taken is decided at every router so it is never defined when the packet is initially sent. It isn't always the most logical route taken as it's normally decided by the network traffic at the time.

2.8.2. Asynchronous Transfer Mode (ATM)

ATM is often used to route telephone calls. Unlike IP the routing relies on a predefined route being set up by a controller when the call is established. Every cell is routed along

the same route which means cells are more likely to get to their final destination at the same time. The disadvantage is any hardware failure along the route will result in the connection being lost.

3. Method

This chapter describes how the project will be conducted.

3.1. Development Tools

MPLAB is the free Integrated Development Environment (IDE) offered by Microchip for the development of embedded applications employing Microchip's PIC Microcontrollers. MPLAB IDE runs as a Microsoft Windows application and includes free software components for fast application development. The IDE also includes a hardware simulator so applications can be debugged before deployment. The PICKit 2 is a USB programmer for baseline and midrange 8-bit and 16-bit microcontrollers.

3.2. Choice of Programming Language

Prior to PIC18 MCU the assembly language was the recommended language when developing firmware for microcontrollers as described by Bates (2004). This was mainly due to the restrictive resources available to the developer resulting in the need to develop in the most code-efficient manner.

The PIC18 MCU series is the most powerful midrange microcontrollers available. Generally having larger memory sizes and a 75 16-bit instruction set, the C programming language can be used for firmware development. It has to be noted that firmware written in C will still not be as code-efficient as when it is written in Assembly, so the resources gained by using PIC18 may not result in extra performance. Code optimisation is still important and the amount of extra performance will rely on the application and how it is structured. For many developers, C is still a baseline language to learn and use and as it is a high level language, human readable design can easily be implemented. The MPLAB C18 compiler provides useful libraries for setup and I/O handling which supports the developer to concentrate on program logic rather than low level register toggling.

The ultimate factor for the choice of programming language for this project was the developer's experience with the two languages. C was chosen as the programming language for this project as the PIC18 microcontrollers target C development and the developer's experience was far higher with this language.

3.3. Chosen Software Process Model

An overview of the different software process models are detailed in the Appendix A. This project will follow the evolutionary development approach as it allows for changes

to occur during development. With the timescale available it can be assumed that the requirements and goals of the project will change. The understanding of the problem domain and solution system will be constantly changing over time and the use of throw-away prototyping will gain understanding of these areas.

3.4. Design Approach

The design of the system will rely heavily on prototyping work carried out prior to this project commencing. A diary of this work can be found in Appendix I but will not be detailed further. As with the theme to this project design abstraction can occur at many levels in both hardware and software of this project and the following selections detail both the higher and lower levels of design abstraction. The design of the system will be conducted to meet all requirements laid down in the Appendix B.

3.5. Developer Best Practice

Source control software has been used to management increments to changes of source code and documents. It is developer best practice to check in revisions of source code at project mile stones so a history is formed of the source code and development. Creating a history means that if defects appear during the build or test, code can be rolled back or a post mortem can conclude where the defect was injected. For this project, code was checked in daily or when a major function was completed.

There are many different types of source control software however they will not be reviewed here as it falls out of the scope of this report. For this project Microsoft Visual SourceSafe has been used as the source control software. For good backup practices the database part of SourceSafe, which acts as the server part, has been installed on a remote Windows server away from the development workstation. This installation will mean there will always be two copies of the source code preventing any loss. The choice of SourceSafe has come down to the ease of use in both installation and general use.

4. Requirements Analysis

This chapter details the methods used to form the requirements document. The document can be found in Appendix B.

4.1. Requirements Gathering

Jackson (1995) identifies that there are a defined set of problem types and that any problem should fall into one or more of these types. A problem frame visually models the different problem types which separates the requirements from the problem domain properties. It is hoped that different information is collected and recorded after the problem type(s) is identified and this determines a set of processes to take to aid the development of the requirements document.

Problem domain oriented analysis (PDOA) which the problem frame forms part of has been used to form the requirements for this project. The current system manually controls a model railway where there is no feedback apart from changes to the problem domain itself. The Control Frame can describe the problem domain of this project. Bray (2002) describes some information sources that can be considered good for elicitation such as clients, other similar products and potential users. This project doesn't have a true client but as it solves a real world problem it would have potential customers and users if manufactured. These potential users such as model railway enthusiasts and also the researcher's own experience with model railways formed the main elicitation sources.

4.2. Elicitation from End Users

Informal interviews were held at Pecorama model railway exhibition centre with railway enthusiasts. Two common areas were highlighted by a number of the interviewees. The first was the price of the system. They were concerned that such a system would be expensive and therefore would disregard its target audience. They were asked what they thought was a fair price and the average of all the answers formed R2 of the requirements document. The second area that was highlighted was the functionality of the user interface. The user interface is part of the workstation software and as this software isn't intended as a deliverable the interviewee's discussion about the interface fell out of the scope of the project. Their comments however will be considered to create a future proof architecture. Their main concern was how much control was being automated as they wanted to still be able to control a single train while the system

controlled the rest of the layout. This train driver concept should be applied to future builds that implement a user interface.

4.3. Elicitation from Pre-existing Technologies

Pre-existing technologies are discussed in the background research of this project. The features of the Lego Mindstorms should be considered for this project. The ability to use it for many different Lego models gives it a large degree of flexibility which is perfect for its control domain. It does this by providing the user with a reasonable amount of I/O lines and allows the user to program the RCX using a visual orientated programming style intended for any type of user. Requirements 15 and 19 have been formed from this elicitation. R15 describes the solution system should be scalable because Lego MindStorms success is contributed to its ability to be tailored for lots of different model sizes and complexities. R19 describes the target age of 8+. Lego Mindstorms targets this age also and its setup and programming features are to a level that it is not restricting to what can be done with it so it is the correct target age for this project too.

4.4. Creation of Test Speciation

Creating the requirement document means the requirements can be used to create test cases to be used during both the unit testing and integration testing stages. The tests and results can be found in the Appendix D.

5. Design Specification

5.1. High Level Design

A Block Controller is in charge of a section of the railway layout called a block. The block consists of the control of points, signals and a train. A layout can consist of many blocks with all the Block Controllers being able to communicate with each other. When a train needs to move into a neighbouring block, a request is made to the associated Block Controller for this to happen. When the request is granted the train moves into the neighbouring block along with its details such as destination. This continues to happen until the train arrives at its destination. See Appendix C 1.1.

5.2. The Demo Layout

Two identical mirrored terminating blocks were used. The layout had to satisfy requirements by including both points and signals. Due to the portability constraint a small layout was built which exemplified a large layout, in that it has signals and points as larger layouts do. The use of one point was a given, meaning that the design would incorporate one track branching to form two railway lines. Both of these lines have their own dual colour signal. To create some movement complexity an additional siding is included with two extra points. Single ended trains can 'rap around' by using this siding, which is the act of a train moving from one end of a coach to the opposite end. The size of the layout is equivalent to three railway lines wide and the blocks can be parted from each other making the layout perfectly portable as stated in the requirements. See Appendix C 1.2.

5.3. Train Movement

When controlling a model train we are effectively controlling a low powered DC motor. Motor control can be more demanding because the control device, the motor, has its own characteristic which has to be taken into account when designing both the hardware and software that will control it.

Motor control using microcontrollers are extensively documented within the electronics community so this will be taken as a given and the intent of this project isn't to improve on any chosen approach.

To control a motor the two pulse width modulation (PWM) channels will be used. Each channel will control a different direction of movement. The PWM is a useful method of

simulating an analogue output to produce a proportional drive current from a digital device. The microcontroller chosen for this project has PWM built into the hardware and this will be used.

PWM on its own can not generate the required current needed to move a model train at full speed. The microcontroller uses a 5 Volt supply and the PWM also uses the same voltage therefore this is the maximum it can supply to a motor. The use of an H-bridge overcomes this problem which can be thought of as an electronic switch. Using a separate 12 Volt power supply which will be used only to power the train, the output from the PWM can be fed into the H-bridge acting as the switching factor. Every time the PWM switches the 12 Volt supply will also switch. The job of an H-bridge doesn't stop there however as its main use is to control the direction of travel of the motor. Made of four switches depending on the state of the switches the H-bridge can reverse the polarity of the motor changing its direction. Another function that will not be used in this project is a state that will 'brake' a motor. This is idea in motor control situations when moving heavy loads that could easily move once the power is removed.

The makeup of a model train has to be considered when designing the control technology. It has to be noted that a model of a train differs considerably in size and weight from its full scale counterpart. Its movement characteristic also differs and behaviours that we regard as a result of the laws of physics on a full scale train may have to be artificially recreated for its model. When removing the power from a full scale train it will gradually come to a halt overtime. Removing the power from a model train however will result in an unrealistic jolt stop. Acceleration and deceleration will therefore have to be recreated in software to make the movements of the trains realistic.

5.4. Communications

The chosen microcontroller has both USART and USB communication hardware available. USART is simpler to use but doesn't have any of the more advanced features of USB, namely the addressing system and the speed.

The USART was used as it would require little understanding to achieve simple communications between microcontrollers. The microcontroller has only one USART with pins for transmitting and receiving data. If both pins were connected to the same node, for example another microcontroller, it would be impossible to communicate with a third node, being for example a computer. The communication design therefore follows a ring architecture (Figure 1.1) where one node's transmit pin goes to another

node's receive pin. The limitation to this is any message from one node may need to be send via a middle node creating the need for a relay protocol.

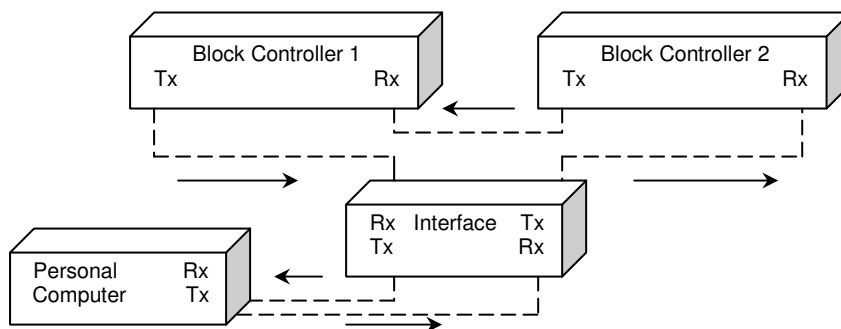


Figure 1.1

The communications protocol created shown in Appendix C 1.3 and 1.4 details that every message sent from a microcontroller must have a 'to' and 'from' token to determine where the message is going. If a node receives a message that isn't intended for it, the node will relay the message to the next node along the ring. The weakness of this design is if communications breaks between one of the nodes then the whole ring is compromised.

Communication is actually achieved via ASCII characters using a XML-like notation. It was decided to not implement full XML as this would be overkill for what it was needed for and also the resources of the microcontroller available. The XML-like notation can be easily parsed which satisfies the requirements.

The main principal of the communications protocol is that a request is made to send a train into another neighbouring block. With this request information is also sent about the train's type and the train's destination. A reply is made back either granting or rejecting the request.

5.5. Managing Points

Due to the operating voltage of points that ranges from 12 to 16v DC it is not possible to connect a point directly to a microcontroller as the output from any I/O line is only 5v. A Darlington Driver IC is used which can control three points, as each point uses two I/O lines. The points are supplied with a separate 16v power supply and the driver can be seen as a switch. When the output of an I/O line goes high this is relayed by the driver to the separate 16v circuit moving the point in the desired direction.

This configuration does not fulfil the requirement of preventing point burn out in a malfunction situation. A Capacitor Discharged Unit (CDU) was fitted between the 16v

supply and the points. Its purpose is to only supply a short burst of electricity to a point when the circuit is opened, determined by the Darlington Driver. After it has discharged it will not recharge until the circuit is closed again. The CDU prevents point motor burnout because regardless of the time the circuit is open the same burst of power is supplied to the point.

5.6. Managing Signals

The signals are simply Light Emitting Diodes (LED). LEDs only require their own resistor to take the output current from the microcontroller of 5v down to around 1.2v. They also require their own I/O pin from the microcontroller. As Red (Danger) and Green (Clear) are the only LEDs to be used, each signal will cost two I/O pins from the microcontroller. It is possible to multiplex the I/O pin so only one pin is required but this will require additional interfacing circuitry and will create upgrade limitations so hasn't been used.

5.7. Managing Opto Switches

Opto switches have been chosen as the sensor to the system as they don't require anything being placed on the underside of the trains. Five will be used at different places on the layout (See Appendix C 1.2) and will be mounted in the track facing up. They are formed of two parts, an infrared LED which can be powered like any LED and the sensing part that will be connected to a microcontroller pin. The pin the sensor is connected to is kept high by another supply and when the sensor detects the infrared light it grounds the pin sending it low. To change the sensitivity of the sensor the resistor to the other supply can be changed.

5.8. Selecting a Microcontroller

To efficiently program using the C programming language a PIC18 microcontroller was used. To handle all the hardware I/O a minimum of 18 lines were needed. The Microchip website details and compares the specs of all the microcontrollers in the range. To have at least 18 I/O lines the website detailed that a microcontroller with 28 pins would have to be used. The PIC18F2550 was chosen as it had the fastest CPU speed of 48MHz and also the largest amount of program memory of 32 Kbytes. This microcontroller also has all the communications and PWM hardware required as well as an additional 6 I/O lines however some of these may be multiplexed with other hardware applications. See Appendix C 2.2 for the pin assignment.

5.9. Block Controller

The Block Controller includes all the hardware to control the signals, points and trains. It is formed around a PIC18F2550 microcontroller being supplied with a 5v power supply. The controller's main power input is a 16v unregulated transformer however this is dropped down to the 5v required with the use of a regulator. The motor control uses a separate 12v supply to prevent brownouts resetting the microcontroller; this however has never been proven true but has been still implemented as a safety measure. Many Block Controllers can share the same power supplies so power and communication cables connect the different controllers together. The H-Bridge and 5v regulator both have a heat sink fitted to them and should be the only components to generate a substantial amount of heat. The regulator constantly generates heat where the H-Bridge only produces heat when it is active and a train is moving. For all the different I/O lines that are fed in and out of the microcontroller, a terminal block is associated with every one of these. A terminal block is a simple means of attaching the cables from the external components to the Block Controller. 'Push to release' terminal blocks have been chosen to prevent the need for a screw driver when securing or removing a cable. The overall layout of the Block Controller has been created with size in mind. All the components have been arranged as close to one another as possible but also considering cable attachment and removal. See Appendix C 2.3 for a schematic.

5.10. Software Design

5.10.1. Overview

The software is based on a Super Loop design as described by Pont (2002). It is divided into two main tasks, the first is to send and receive messages and the second is to manage the movement of the trains. The 'Comms Manager' reads a message, parses it, validates it and then determines whether it is a new message or a reply, see Appendix C 2.5 for a communications receive diagram. The 'Jobs' list stores any outgoing or incoming messages and it is shared between both the Comms Manager and the Movement Manager. If a new message is received the request is added to the Jobs list but if it is a reply the job is searched for in the Jobs list and then updated. For messages that aren't intended for the current node, the message is still added to the Jobs list but the 'not sent' flag is raised so it is resent, see Appendix C 2.6 for a communications send diagram. While there are no messages to receive the Comms Manager searches through the Jobs list for any messages that haven't been sent, then sends any that are found.

Platform state is also shared by both the Comms Manager and Movement Manager. Its purpose is to keep a record of what train is at what platform and whether the train needs to depart.

While no movements are being carried out the Movement Manager has a job of granting any movement requests or adding new ones into the Job list. There are only a certain amount of movement routines and they can be divided into two categories, moving away and moving home. The movement requests made by other Block Controllers are always movements to home where the movements away are always made by the current Block Controller.

When a Movement Manager grants a request in the Job list, a home-routine is then set up and the Movement Manager waits for the trains to hit the first sensor then the routine continues. The same happens with away-routines where the Movement Manager finds a Job in the list that has been granted by a neighbouring Block Controller, however the movement happens away from the block. See Appendix C 2.1 for an overview diagram of the software structure. Appendix C 2.7 defines the movement routines and Appendix C 2.4 defines there choice.

5.10.2. APIs

Firmware development is carried out at a much lower level compared to developing for other platforms. The developer not only has full control of the hardware but also has to manage it. Traditionally developers would rely on the operating system to manage the hardware and APIs are supplied so programs can interface with the hardware via the operating system.

To aid development it was considered useful if some basic functions were created that would act as APIs to the hardware. Functions are created to change points and signals, and change the speed and direction of a train.

The creation of a system clock is another aspect of microcontroller development that isn't considered with desktop development. There are two timers available on the chosen microcontroller but they have different resolutions and can be used for different purposes. The timer is setup during initialisation and when it overflows causes a system interrupt which increments a counter. The timer overflows four times a second and the program logic calculates how many minutes and seconds have passed. The timer is used to determine a time for which the points I/O should stay high to open or close a point

and also is used to create delays between certain movement routines. Again API-like functions have been created so interfacing with the timer is easier.

It has to be noted that the creation of program functions does cost memory but this has to be weighted up against the ease of development that they create.

5.10.3. Multitasking

With no operating system there is no easy way to multitask and as a result, this limitation had to be considered when designing the software. Using high level design the software was broken down into major tasks, they are communications and movements. These two tasks are made into two major functions that are polled rapidly to simulate a multitasking environment. Once one function returns the other is called which as long as the functions return quickly communications and movements seem to happen at the same time. There is however always situations where one task may take longer than another and therefore will unfairly hog more processor time. The design uses non-blocking functions to make sure tasks get close to even amounts of processor time. Non-blocking functions return to the caller even when they haven't completed. It is the job of the caller to check whether the task is completed and where the function hasn't completed the function is polled again.

5.10.4. State Machine

The core software design uses an event-driven finite state machine. Each area of the software has defined states, for example reading messages goes from getting the message, parsing it, storing it and then deciding what to do with it. There is no global state but instead many states for the individual components of the software. At the start of each source file a new type definition is declared which defines the individual states of that file. A new instance of that type is created and this is made static so its value isn't lost. The state machine is then formed by a switch statement of the valid states and a initial state declared. When the function that contains the state machine is called only the code that is related to the current state will execute. Each state is responsible for changing the current state and forgetting to do this will put the program into a constant loop.

5.11. Operator Control

Using the 'SET' command of the communications protocol an operator is required to set the train type of any trains present at the platforms. Operators are also able to set the destination of a train and depart them. The detailing of the application used to send

commands to a Block Controller falls out of the scope of this project. At the simplest level a basic serial communications program can be used like the hyper terminal, however the program must relay and resend any messages that are not intended for it. The workstation at which the commands are sent from is known within this project as the 'Scheduler PC' as it determines when the trains will depart.

6. Implementation

6.1. Overview

The implementation of the firmware was broken down into many files, each file representing a major aspect of the system. Main is contained in Main.c, and polls Comm.c and Movements.c. Communications is broken down again into CommsSend.c and CommsReceive.c. There are five distinct movement routines and they each have their own file. Points, Motor Control, Signals and the Jobs list also have their own file. Using multiple files does cost memory however the practice is useful because program design can be easily transferred directly into them. Multiple files can speed up development and it reduces the time it takes to debug software as it assists clarity. Most of the core files contain their own state machine.

The full list of source code and compiled hex files can be found on the accompanying CDROM. See Appendix J.

6.2. Implementation Issues

It has to be stressed that the following issues, while only taking a few sentences to describe, have taken a considerable time to debug. The nature of a microcontroller means there is little or no output to the developer resulting in 'silly' issues becoming time consuming to debug. The following highlights the implementation issues.

6.2.1. Power Supply Issue

Initial trials suggested that the power supply used wasn't up to the job and the train moved slowly. The amp rating was doubled to 1200mA which solved the problem.

6.2.2. H-Bridge Temperature

After running for approximately five minutes the H-Bridge unsoldered itself due to the excessive temperature it was generating. After fitting a heavy duty heat sink to the H-Bridge the temperature was controlled. A heat sink was also fitted to the 5v regulator for good measure.

6.2.3. Darlington Driver Failure

Following use of the points the Darlington Driver which operate the points failed. It was discovered that while one amp was being supplied the maximum that should be supplied

to the driver was 0.5 amps. An additional Driver was added in parallel to solve the problem.

6.2.4. Low Voltage Programming

The Low Voltage Programming was causing any LEDs to flicker. It was found that this was resetting the microcontroller very quickly so this was disabled.

6.2.5. Configuration Bits

The 'config bits' were not exporting from MPLAB IDE meaning the microcontroller was not being setup properly. This was developer error and was resolved promptly.

6.2.6. Master Clear

The Master Clear Line was enabled which was retting the chip. This was disabled.

6.2.7. PIC to PIC Communications

The communications was scrabbled between microcontrollers but there was no problem when using the development board. The microcontrollers must share a common ground to achieved communication, it was found that this was the problem and a ground cable was added.

6.2.8. Power Supply Issue II

Both Block Controllers share the same power supply. As both initialisation sequences happen simultaneously, both points on each Block Controller got closed at the same time. It was noticed that only one point would move as there was not enough power for both. This could also happen at any time if both Block Controllers moved a point at the same time. A Capacitor Discharge Unit (CDU) was fitted to prevent this from happening. The CDU would act as a buffer and discharge when the point is moved. The CDU also has an additional benefit as it prevents a point motor burning out if a malfunction occurs.

6.2.9. Memory Issues

Parsing the communication messages has been carried out by the C function strtok(). While it does a good job of parsing the messages it however makes whatever message was passed into it unusable. There are situations where the received message isn't intended for the current Block Controller, in this situation the message has to be resent but as it is parsed it has been made unusable. Initially the message was copied prior to

parsing so a copy could be used for resending however there wasn't enough memory available and it seemed very inefficient. The solution was to still allow any message not intended for the current Block Controller to be added to the Jobs list but flag that it needed to be resent. The solution created additional work and is still not ideal however compromises have to exist when developing for microcontrollers.

6.2.10. Memory Issues II

Storing characters is the area that takes up the most amount of memory and started to become a general issue. There were situations where the stack was being overloaded caused by the software design preventing global variables. Preventing globals is a common desktop programming practice but it became clear very quickly that having to load private-static variables within a function costs stack memory and while total memory is a respectable size, stack memory is still fairly restricting.

7. Testing

This chapter gives an overview of particular testing strategies and then the strategy used for this project is detailed. Tests and results can be seen in Appendix D.

7.1. Unit Testing

Good software design will result in a modular, reusable architecture that can be easily seen as units of code. These units have a requirement of what can be passed into them and what will hopefully return. What should return should be defined by the design specification and this is compared with the actually result. The more the area of testing is minimised the greater the chance is for identifying defects and where they are located in the code. Constant developer unit testing is always encouraged which may not be recorded however the developer may aid the creation of test cases for future recorded unit testing carried out independently by the test team.

7.2. Integration Testing

Once the units of the system are constructed to form the final product, this again could present an area for defect injection. Tolerances between units may not be compatible, for example a log-in dialogue unit may timeout waiting for the user authorisation unit to respond. It is at this stage where integration testing is carried out. They test the system at a higher level which is determined by the design specification. Daily builds are often used as a form of integration testing but they do not test functionality. They are a test to see if the program will link and compile and may include some general 'smoke' tests but will not test the final build to the same level as when test cases are used that have been derived from the design specification.

7.3. Black Box Testing

The black box testing strategy views the area being tested as a black box where the test is completely unconcerned with the internal behaviour of that area. The area can be either a program function or a complete Block Controller. It is only concerned with the output of the test and the test cases are determined by the design specification.

7.4. Strategy

Myers (2004) describes testing as the process of executing a program with the intent of finding errors. When testing we must keep this in mind as it would be easy to think that

the goal was to find no defects. Testing is a destructive process which explains why it is generally done badly as it goes against our constructive nature. To enforce this further we must define the words “successful” and “unsuccessful” in regards to the test cases. Successful is when a defect is found which may seem the wrong way around but finding defects is what we want to achieve. Testing that concludes with found defects can hardly be considered unsuccessful and the time spent was well invested.

As with any good software development this project uses a number of different testing techniques. All testing is black box. White box hasn't been used for this project as there isn't a safety critical requirement and exhaustively testing the inner workings wouldn't be possible with the timescale given. Unit testing has been used for all the functions created. These tests haven't been recorded and in industry it is usual to expect a developer to constantly test their work at regular stages. A daily build was carried out on days that any development occurred. This project is modular and the Block Controllers form individual modules of the system and have been unit tested. All the modules were then linked together and integration tests were carried out which forms tests 1 to 38 of the test plan. The tests carried out were formed with the use of the requirements documents and also the design specification.

7.5. Defects

Tests 31-38 successfully failed. The test description is:

"Conflict Movement Routines: The train at the platform 'P' of Block Controller 'BC' has had its destination 'D' set which includes a destination platform 'DP'. The train-type has been set a double ended train and destination platform 'DP' is NOT clear of any other trains. It has been issued with a depart command."

7.6. Post Mortem

A group of tests with the same description failed integration testing. Test 31 to 38 tried to create some real world conflicts of movements within a block. It highlights not an implementation defect but a design limitation. The tests create a lockup situation that can be resolved but it requires the movement of the blocking train. This could result in a total lockup of a larger layout if many trains were waiting for preceding trains to move. Improvements to user feedback and movement logic will have to be considered in future builds to solve this issue.

8. Evaluation

This chapter gives an evaluation of all the major selections of this project.

8.1. Background Research Evaluation

The background research gave the project a good foundation to what control systems and techniques are currently available. The solution system created by this project is based around many of the findings of the research. The review of Lego Mindstorms aided the user focused portion of the project and highlighted what features a home user control device comprised of. Researching the different sensor types and what they are normally used for in control situations helped specify which sensor to use. The research into the different types of microcontrollers encouraged prototyping and proof of concept which came in useful when deciding which microcontroller to use for the Block Controllers. Finding out how full scale railways handle control eased the creation of a partial solution and this coupled with theories behind computer networking completed the solution.

8.2. Design Evaluation

The overall design fulfils the requirements stated in the requirements document. There are however some limitations the design has created and one was highlighted during system testing.

8.2.1. Feedback

Feedback from the Block Controllers is limited to both the rest of the system and the users. The user is unsure if the train-type or destination has not been set correctly nor do they know if the depart command has been received correctly. The system has no feedback of whether job requests have been received correctly either. Once a message has been sent it is not resent again. Any break down in communications between any of the system modules could result in a lock up because a Block Controller is waiting for another to reply. This is highlighted by tests 31 to 38 which demonstrated the lack of transaction complexity between the Block Controllers.

8.2.2. Communications

The communication protocol uses a XML-like syntax. The syntax includes a lot of duplication, for example `<to>` and `</to>`, which has to be stored by a limited resourced

microcontroller. During implementation some memory problems occurred and the syntax was highlighted as an issue. The messages will never need to be read by humans so a smaller more efficient syntax could easily be designed. The improvements could still allow human orientated debugging but translatable code-based syntax could be used. The communications medium is an area of design weakness. The ring architecture means that any weakness in the relay mechanism whether physical or software based will take down the whole system. Physically little can be done to prevent a physical break down apart from specifying a different communications medium like USB. Software though could check to see if the connection is active which it currently doesn't do. A Block Controller could issue a message to itself and if it receives the message it would know the connection is healthy.

After not knowing whether messages have been received the Block Controller appends no age to the sent message and they wait in the Job list until replied to. If a reply is never received a design limitation appears. Improvements should be made so any message that hasn't been replied to within a certain time should be resent. This also means that replies would have to be sent more than once too.

8.2.3. Movement Logic

The communication protocol specifies a destination platform but it was felt unrealistic to what happens with full scale railways. A train travelling from Dundee to Bournemouth wouldn't know what platform at Bournemouth it would eventually stop at. During the journey anything could happen which could causes delays to preceding services resulting in journey alterations. To state which platform to arrive at creates service limitations as Block Controllers will reject jobs if the platform is already taken. It should be the Block Controller's responsibility to determine which platform to send a train to so the decision can be made at the last stage of the journey.

8.2.4. Development and Upgrades

To upgrade the Block Controller's firmware the microcontroller currently has to be removed from the Block Controller and placed on a development board to then be connected to the PICKit 2 programmer. It would have speeded up development if in-circuit programming pins were included which would allow the microcontroller to directly interface with the programmer while connected to the Block Controller. This should be considered for future builds.

8.3. Implementation Evaluation

8.3.1. Communications

There have been a few features of the communications protocol that wasn't implemented so the project could be delivered on time. The Get functionality wasn't implemented which allowed the desktop computer to retrieve the locations of the trains. Another feature that wasn't implemented was the Speed option. Both features don't limit the total functionality of the project as the speed is set to a constant value and the get function can be seen as an extra.

8.3.2. Possible Lockups

The overall design of the code is a good one. Using an event driven finite state machine has meant the system is prevented from carrying out operations it shouldn't do. It has however created possible areas for system lockups. The events for the system are generally inputs from the sensors meaning as long as the external hardware acts in the way the system expects everything will run smoothly. This of course doesn't always happen and in most cases the system will wait forever until an external state changes. These possible lockups could be easily removed by the use of timeouts and recovery routines. For example once a Block Controller grants a request it sets up the movement routine and waits for the train to enter its block. If the train never appears the system hangs forever but if a timeout was implemented it could mature and the system could presume the reply it sent was never received and could resend it.

8.3.3. Over Engineered

Future proofing designs is a goal when developing software. To do this we tend to over engineer a solution and worry about 'what if' situations. We also design to prevent restrictions later but this can create unnecessary complexity. The implementation of this project does suffer from a degree of this. There are two system types that were thought should be different but they turned out to be basically the same type. The 'to' and 'from' values are different from the 'destination' however it seems now they could have been the same type which has resulted in some unnecessary conversions during runtime.

8.4. Project Plan and Timeline

The predicted timeline can be found in Appendix G where the actual can be found in Appendix H. Prototyping and a proof of concept stage were carried out prior to the start

of the core project. It was delayed on a number of occasions due to other academic commitments. The project plan found in Appendix F has been altered since it was first submitted as the workstation software was dropped from the deliverables of this project. While it is a simple serial program, it would have required a separate development lifecycle and would require additional time set aside to develop it. With some of the design for the Block Controller already not implemented due to time constraints it was impractical to set aside anymore time for the workstation software.

The creation of a diary of events helped the write up considerably and also defended where the time was used. This idea was taken from the researcher's industry experience where developers are encouraged to record their thoughts and ideas in a log book for intellectual property reasons.

The actual project timeline is nothing like the one created at the start of the project. This is no surprise however because from week to week the time committed to the project was different due to other academic studies. This was known from the start but little could be done to make the predicted timeline any accurate. What would have improved project management is if revisions of the timeline were created at regular intervals the project progress could have been evaluated more often and hopefully speeded up. Over all, the project was delivered with all major functionality and on time which is the main requirement.

8.5. Further Developments

The modular design of this project means that new functionality can be added at any time. The current functionality would satisfy only the basic of the railway enthusiast requirements. The majority though don't have basic requirements and would want more functionality if this project was made into a product. Before looking at further developments it would be sensible to remove the design constraints highlighted in the evaluation chapter. The following describes only a selection of the further developments that could be made.

8.5.1. Expand I/O Capability

With the current configuration only three points, two signals and one train can be controlled by a Block Controller which is a big limitation. There are two ways this could be achieved; Firstly the PIC18F2550 could be replaced with a 4550 which is the same microcontroller type but has 40 pins rather than 28. With the extra pins comes additional ports D and E and the ability to attach more points and signals onto a single

microcontroller. The second way is to pair up many microcontrollers and create an master-slave architecture where the master acts as the brains telling the dumb slaves when to set their pins high or low.

8.5.2. Level Crossing Block Controller

A Level Crossing Block Controller would be based on a normal Block Controller however it would have additional logic to operate the barriers and the warning lights of the crossing.

8.5.3. User Speed Control

It may be seen by some that the current control of 'Set & Go' doesn't allow the users to control the speed of a train anymore which is something that they enjoyed. It would be possible to give back this level of control back to the user by issuing commands from the workstation software.

8.5.4. Implementation of USB

The USB pins of the microcontroller were on purposely left free so it could be implemented at a later date. Using USB would eliminate the weakness in the communications mechanism as the relay feature could be removed.

8.5.5. Information Aids

As found with full scale railways, service announcements could be broadcasted with the use of the desktop computer to show when trains were arriving or running late. Also a arrives and departures display could be created using cheap LCD displays and a microcontroller as a driver. This could display either system wide information or be tailored to a station. It would give a layout a very realistic feeling and feed more information back to the user which currently is lacking.

8.6. Evaluation of Professionalism

The development of this project considered the British Computer Society's Code of Conduct and Code of Good Practice. The developer of this project is a member of the BCS and is required to follow these codes to continue the membership. Examples of the uses of the codes are developing the project with clear deliverables and seeking similar projects to benefit from the lessons learned.

9. Conclusion

The project set out to see if using modular control architecture achieved any benefits in development and the future use of a control system. The project has demonstrated that good modular design is achieved by the modules of the system exhibiting low coupling and high cohesion. The project has proven that this formula provides the following benefits:

Scalable

The ability to add or remove Block Controllers from the communications ring at any time makes the architecture very scalable. The only requirement is the new Block Controller is assigned an address and the routing tables of its neighbouring Block Controllers are updated.

Parallel Development

The lack of interface commitment means each module can be developed at different rates and by different people. This was clearly demonstrated during the implementation stage of this project as actually only one Block Controller was created and tested with the Hyper Terminal at a time.

Clarity of Roles

The result of high cohesion creates clarity of a module role. The role of the Block Controllers is to operate the railway hardware where the desktop PC software has a role of sending commands.

Maintainability

The clarity of roles and lack of interdependencies makes maintenance a simple task to achieve. This is a benefit derived from the other benefits but because maintenance is a key stage on any software lifecycle it is worthwhile emphasising this feature of modular design.

9.1. Summary

The benefits discovered show that this has been a successful project. It is no way ready for production but its design has created a good foundation onto which new functionality can be added. The system will now be taken out of the scope of this project

and onto the next stage which will include the implementation of the further developments that have already been suggested.

Bibliography

- BATES, M., 2004. PIC Microcontrollers: An Introduction to Microelectronics. 2nd ed. Oxford: Elsevier Ltd.
- BATESON, R. N., 2001. Introduction to control system technology. New Jersey: Pearson Education Inc.
- BAUM, D. 2000. Dave Baum's Definitive Guide to Lego Mindstorms. Emeryville: Apress.
- BRAY, I. K., 2002. An Introduction to Requirements Engineering. Harlow: Pearson Education Limited.
- COULOURIS, C. and DOLLIMORE, J. and KINDBERG, T., 2001. Distributed Systems Concepts and Design, 3rd ed. Harlow: Pearson Education Limited.
- CRISP, J. 2004. Introduction to Microprocessors and Microcontrollers, 2nd ed. Oxford: Elsevier Ltd.
- ELLIS, R., 1991. Data Abstraction and Program Design. London: Pitman Publishing.
- GAJSKI, D. and VAHID, F. and NARAYAN, S. and GONG, J. 1994. Specification and Design of Embedded Systems. Englewood Cliffs: P T R Prentice Hall
- HINSON, J. 2008. The Block System[online]. Wealdstone. Available from: <http://www.signalbox.org/block.shtml> [Accessed 3 November 2007].
- MICROCHIP TECHNOLOGY INC, 2005. MPLAB C18 C Compiler User's Guide. Chandler: Microchip, DS51288F.
- MICROCHIP TECHNOLOGY INC, 2007. PIC18F2455/2550/4455/4550 Data Sheet. Chandler: Microchip, DS39632D.
- MYERS, G. J., 2004. The Art of Software Testing. 2nd ed. Hoboken: John Wiley & Sons Inc.
- OLIFER, N. and OLIFER, V., 2006. Computer Networks Principles, Technologies and Protocols for Network Design, Chichester: John Wiley & Son Ltd.
- OPTEK TECHNOLOGY INC, 2006. Long Distance Reflective Switch OPB732. OPB732WZ Datasheet. Carrollton: Optek, Issue A.2.
- PONT, M. J., 2002. Embedded C. Harlow: Addison Wesley.
- PANDA, P. and DUTT, N. and NICOLAU, A. 1999. Memory Issues in Embedded Systems-On-Chip. Norwell: Kluwer Academic Publishers
- SOMMERVILLE, I. 2001. Software Engineering. 6th ed. UK: Addison Wesley.
- VAHID, F. 2002. Embedded System Design. Hoboken: John Wiley & Son Inc.

Appendix A: Software Process Models

To try to structure the process of software engineering, process models exist that try to cater for the different approaches to development. Sommerville (2001) discusses three models: The Waterfall model, Evolutionary development, Formal systems development and Reuse-based development.

The Waterfall model is made up of some well defined phases. The model assumes that development will always be productive and doesn't allow refinement or rework of a phase once the phase has been completed. For the majority of developments this is a disadvantage as unless the requirements and the implementation processes are well understood it is likely that the requirements will change requiring rework. Where these are understood the Waterfall model would be recommended however in practice no projects are fully understood. The requirements gathering process generally uses a customer and it is often difficult for them to state all the requirements explicitly. The many possibilities for change within a development have meant that projects that solely use the Waterfall model have a high failure rate.

Evolutionary development uses an iterative method to develop parts of the project at different times, at different rates and allows for refinement, changes and rework. The initial project is rapidly developed from a draft specification and the known parts. This is then refined with the customer's input and developers' queries. An amount of evolutionary development is found in most modern software projects as they encompass a level of prototyping that all developers carry out, even if not officially. The Evolutionary development model has been used throughout the development of this project.

Formal methods rely on mathematical proof to determine that specifications will be guaranteed to conform to requirements. They aid quality and productivity and are very useful when it comes to implementation as they can be easily transferred to code. Its major drawback however is its perceived not easy to learn and when learnt has a limited audience of who can understand it. The lack of formal method understanding within the industry has led it to it being only used for specialist areas such as military and safety critical systems.

Reuse-based development is used in the majority of software projects. It happens at most stages normally informally when the developer has knowledge of an existing design, theory or code snippet. Reuse is seen as a key element of rapid development as time is

saved not having to reinvent the wheel which is also the main advantage. Its boom has happened in recent years with the aid of the internet and open-source projects releasing code to be used by all. Requirements compromises may occur however, which can lead to delivery of software which does not meet the needs of the customer. Reuse has been used where possible during the development of this project.

Appendix B: Requirements Document

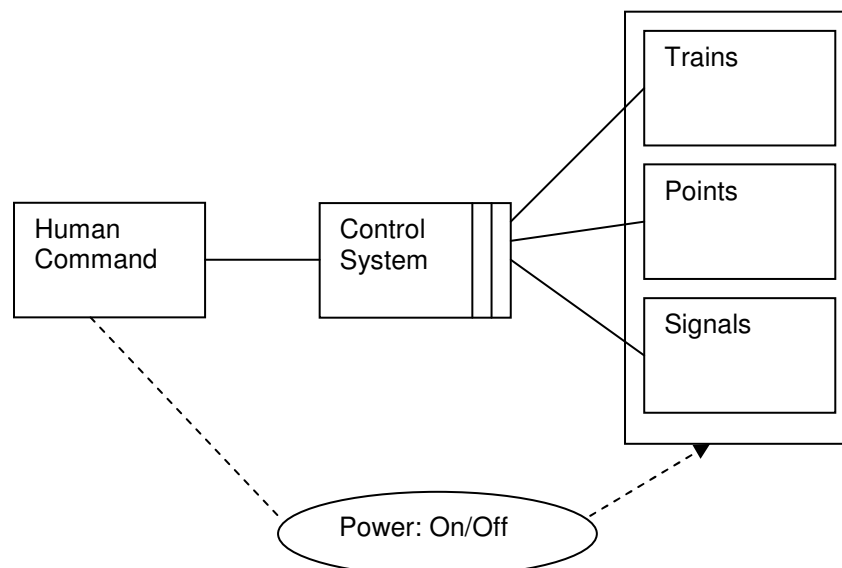
1. Problem domain description

1.1. Introduction

A system is required to automate a model railway layout. The management of train movements should be solely carried out by the system. The system will allow the user to set the destinations of the trains which will allow the trains to depart.

1.2. Problem frame

This is a simple control system as shown here.



1.3. Trains

For a train to operate it requires a voltage equal to or less than 12 and an ampere between 0.5 and 2.0. This drives the train's DC motor which will run at a speed determined by the supplied voltage. Other factors may alter the speed such as motor type and motor age, but these can not be detailed here. The power is supplied via the rails the train rests on. Changing the polarity dictates which direction the train will travel.

1.4. Points

Points have two states, open and close. When in the open state the train's direction is changed normally to another railway line. When in the closed state the train continues in the same direction it was travelling.

1.5. Points Motor

Point motors are formed around two electromagnets and depending which one is energised determines whether the point is opened or closed. To operate they require between 12 and 16 volts at 1 amp using either an AC or DC supply. As the magnets are energised they begin to warm considerably and will eventually burnout. For this reason the manufacture describes that they only require a 'burst' of power which can be considered around 0.5 seconds. The motors require two lines of input, one for each of the electromagnets and they share a common ground.

1.6. Signals

The signals are based around Light Emitting Diodes (LED). There are two colours, Red and Green, which each require their own line of input and they share a common ground. To operate they require 1.2 volts and are forward biased so polarity should be checked. The British Railway Code states Red (Danger) should not be passed by a train unless instructed to by a signalman. With a Green (Clear) signal the train should proceed. Signals should reset to Danger once the majority of the train has passed the signal and both colours should not be operational at the same time.

1.7. Track

The track is the means to supply the power to the trains and it also dictates the direction the trains will travel. The track forms a circuit and should only have one train on it at any one time. Points form locations along the circuit where isolation is possible. Isolating sections of the layout makes it possible to have more than one train on the layout but not actually being powered. More than one circuit can exist on a layout and the track that the individual circuits use can never be shared.

1.8. Movements

Train movement is only possible on track that is being supplied with power. Track can be easily isolated by a point closure and the point state has to be considered when designing train movements.

1.9. Layout

Layouts can be any size and can have any number of powered circuits. They traditionally are a loop design as this is the easiest way to achieve control without automation. They can have any number of points and signals as well as many other railway models.

1.10. Requirements

1.10.1. Commercial Constraints

- R1 The project must be completed by the 28th March 2008
- R2 If sold as a production product the control system shouldn't cost more than £50. to manufacture.

1.10.2. Design Constraints

- R3 The development of this system should use technologies and tools available within the timeframe stated.
- R4 The system must be based around the use of a microcontroller.
- R5 No modifications should occur to the trains, signals or points.
- R6 The design must be modular and scalable.
- R7 Train sensing technologies shouldn't be intrusive to the existing layout.
- R8 A demo layout should be created and must be small and portable.

1.10.3. Functional - Ordinary

- R9 The user must be able to set the destination of a train.
- R10 The user must be able to depart a train.
- R11 The user must be able to set the train type; this must be stored for the total duration the train is located on the layout.
- R12 The system must manage the full management of the train's movement from starting location to destination.

1.10.4. Performance

1.10.5. Speed

- R13 The system should respond to all user commands within 0.5 seconds.
- R14 The time that it takes for a sensor to detect a train and the system to refresh its internal location register should be 0.2 seconds.

1.10.6. Capacity

- R15 The system must be able to control both small and simple layouts with less the 3 points and signals and large and complex layouts with 20 points and signals.
- R16 The system must handle at least one movement train.
- R17 The system must handle at least 2 static trains.

1.10.7. Reliability

R18 The system should be fully operational for at least 8 hours a day.

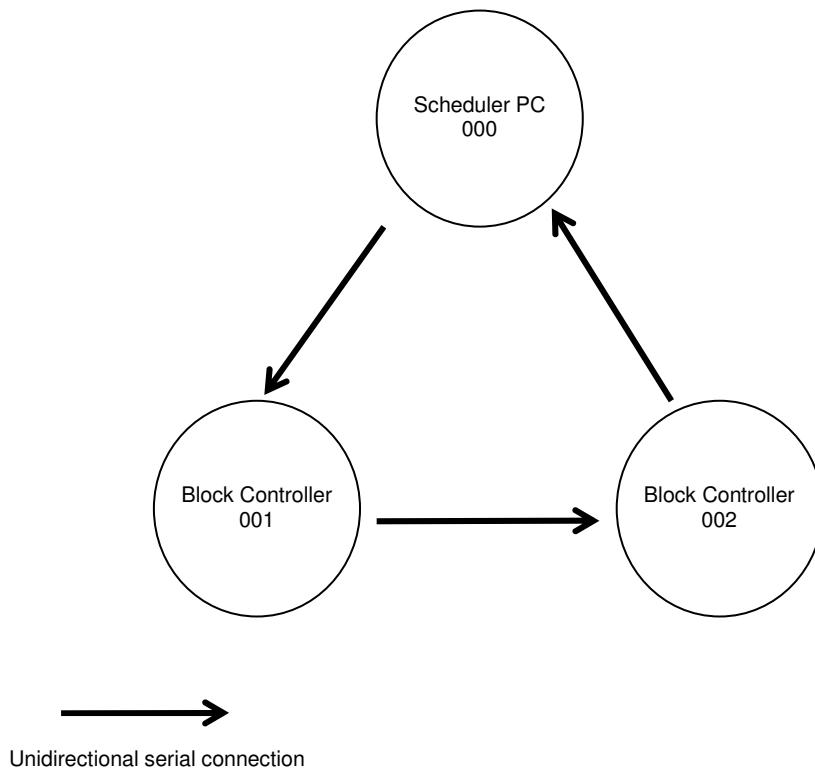
1.10.8. Usability

R19 The system should be targeted at children with the required intellect being equivalent to a person aged 8+.

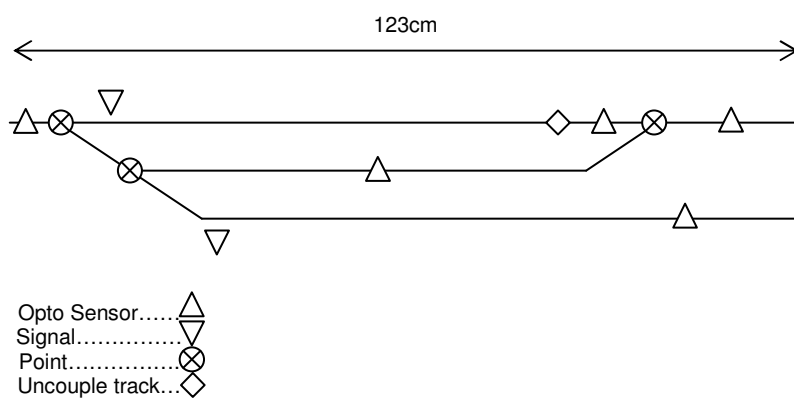
Appendix C: Design Specification

1. System Wide

1.1. High Level Design Dataflow Diagram

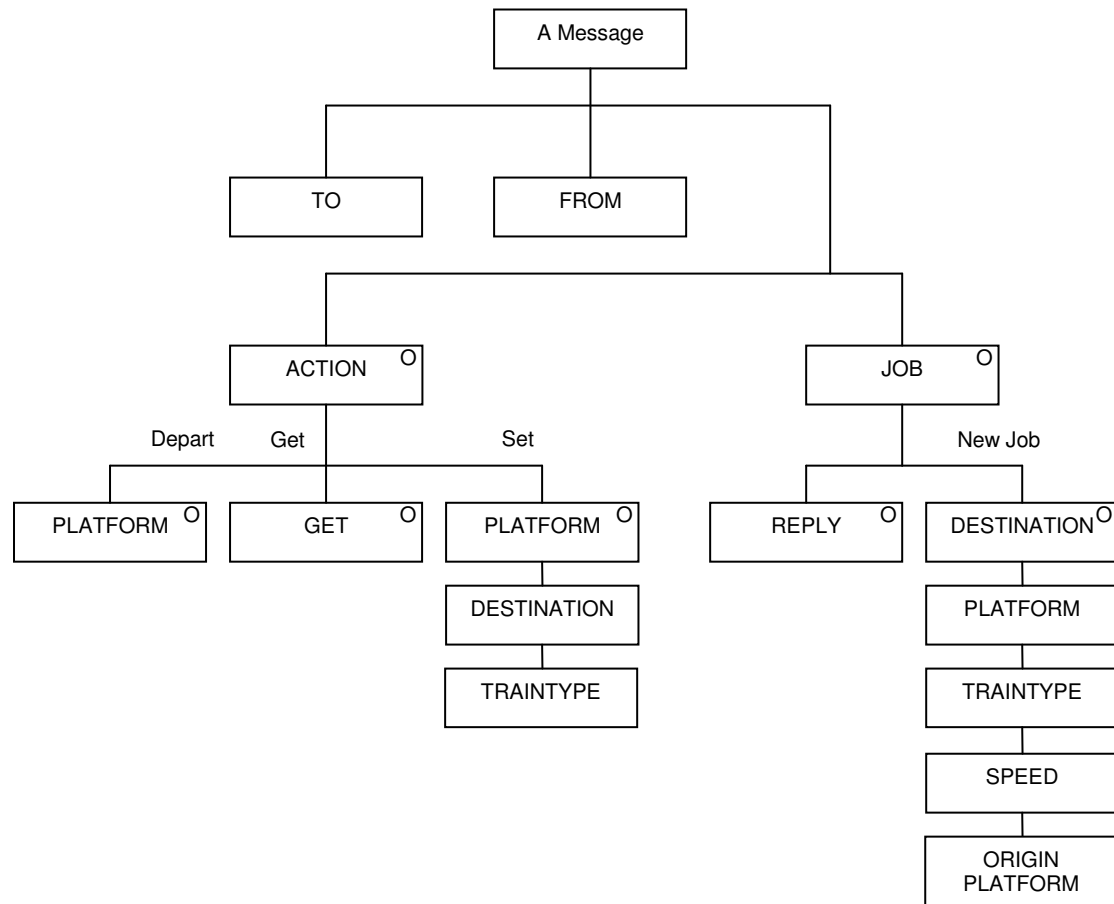


1.2. Railway Track Block Layout



1.3. Communications Protocol

Each box represents a XML-like element



1.4. Communications Backus Naur Form

digit ::= "0" | .. | "9";

letter ::= "a" | .. | "z";

to ::= {digit}³;

from ::= {digit}³;

job ::= {digit}⁴;

action ::= "depart" | "get" | "set";

destination ::= {letter}³;

tratype ::= "emu2" | "emu3" | "emu4" | "br47c1" | "br47";

platform ::= digit;

speed ::= {digit}³;

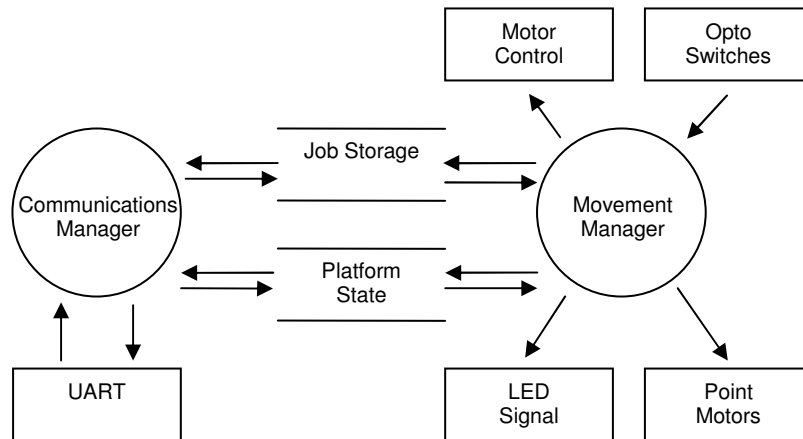
origin ::= digit;

```

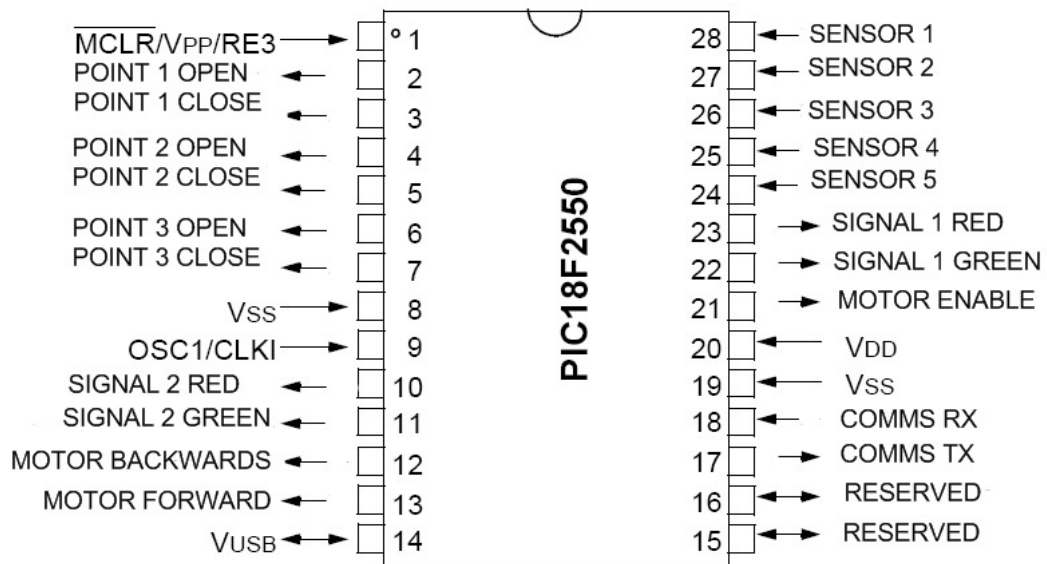
get      ::= digit;
reply   ::= "granted";
    
```

2. Block Controller

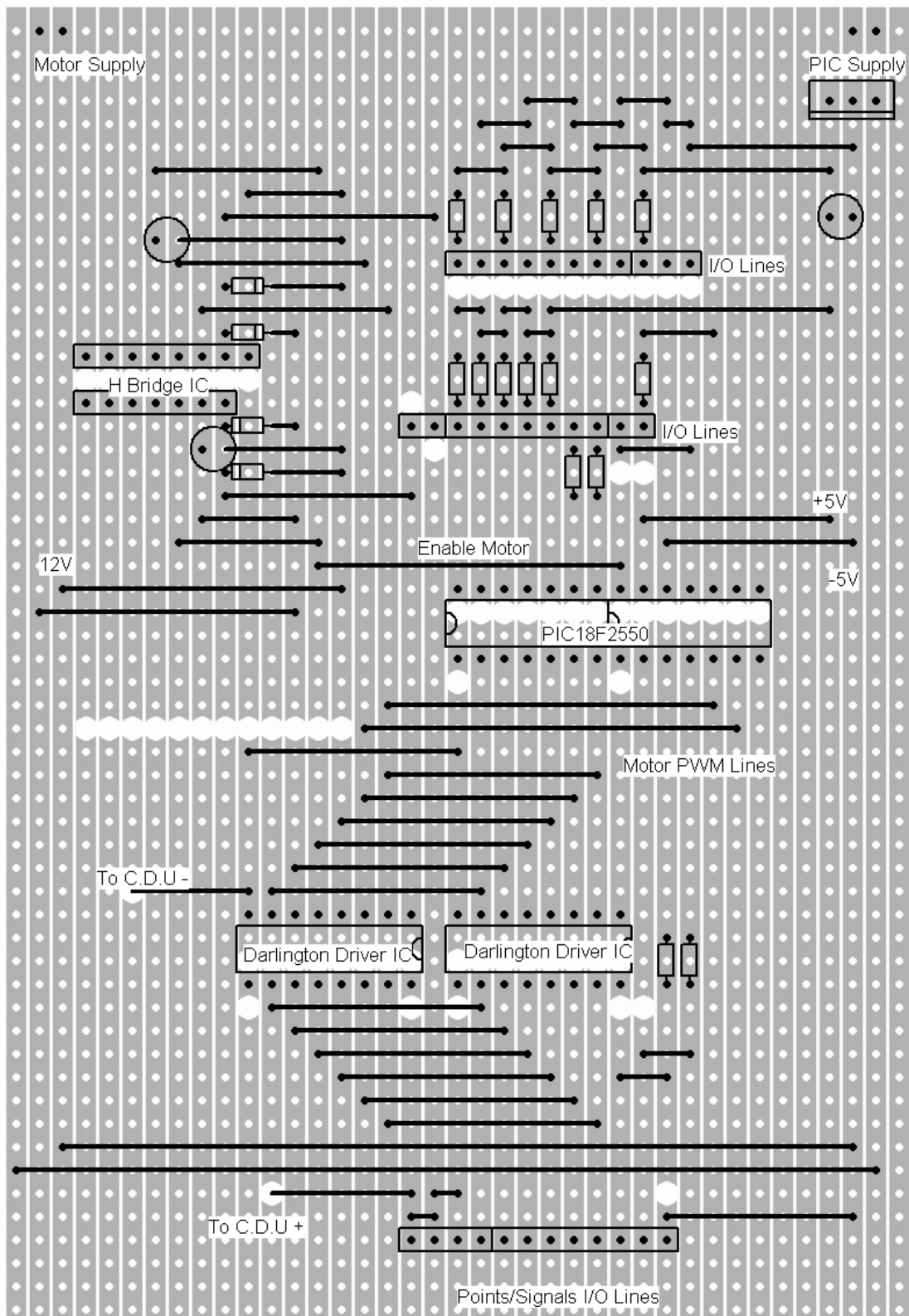
2.1. High Level Design Dataflow Diagram



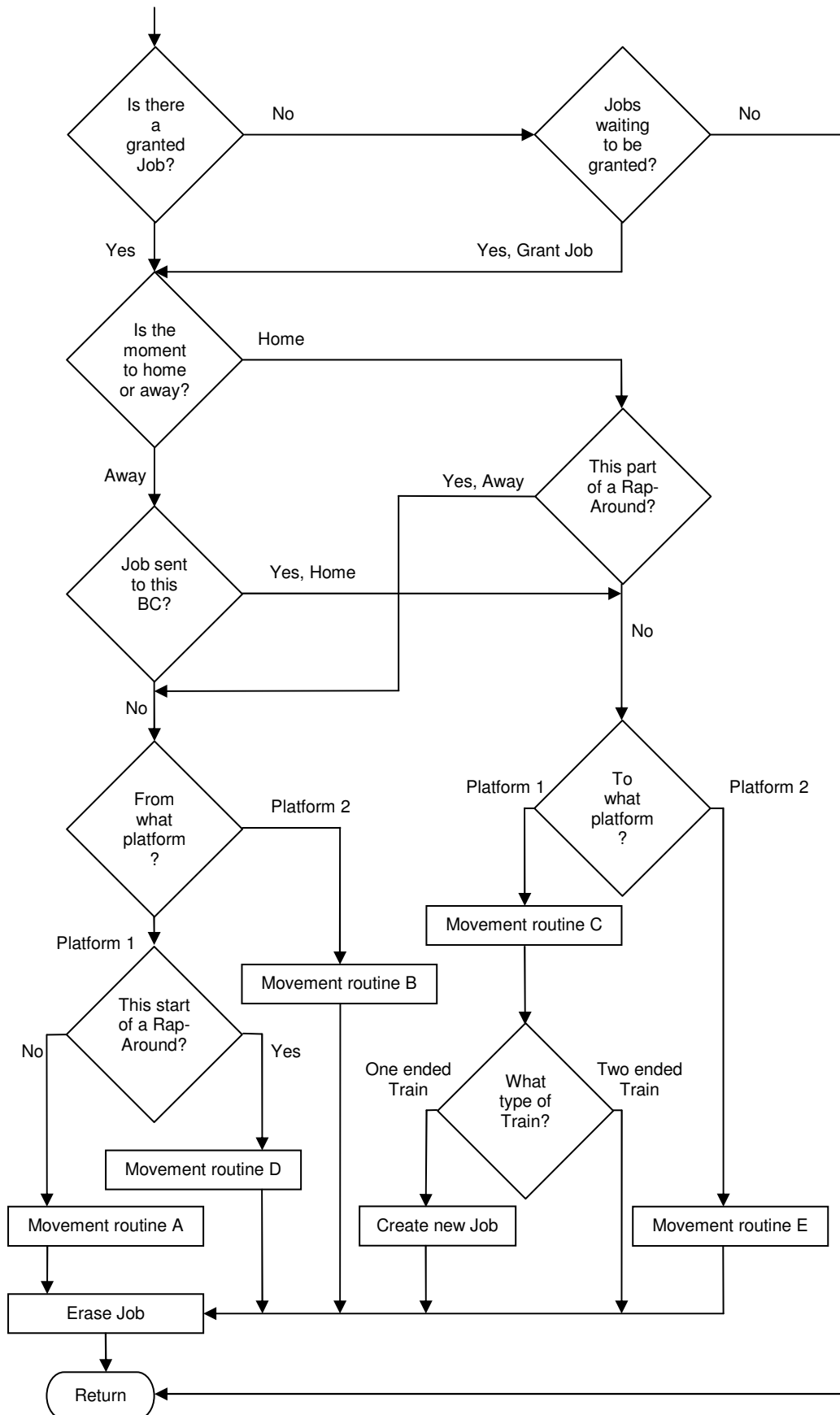
2.2. PIC18F2550 Pin Configuration



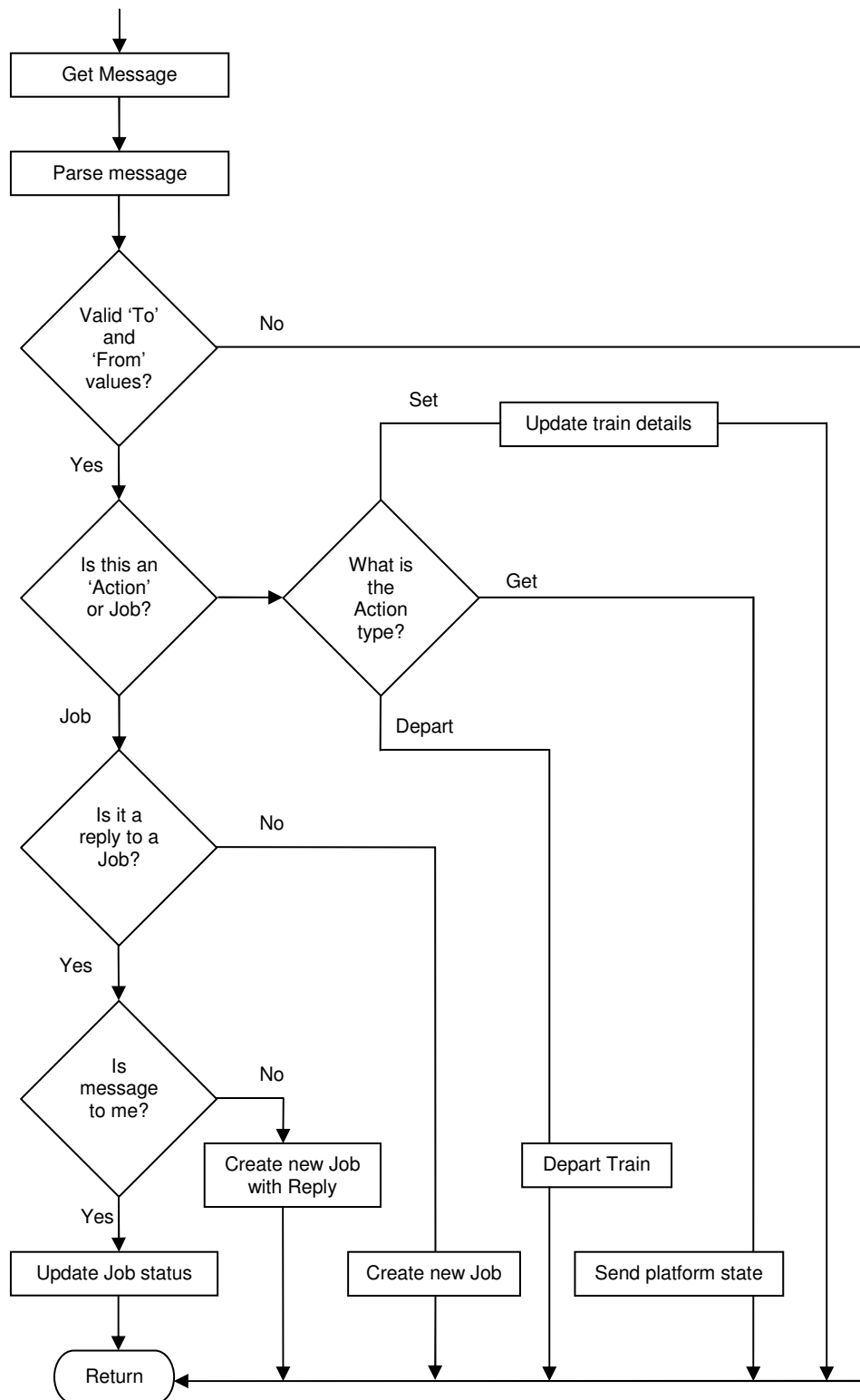
2.3. Schematic Diagram



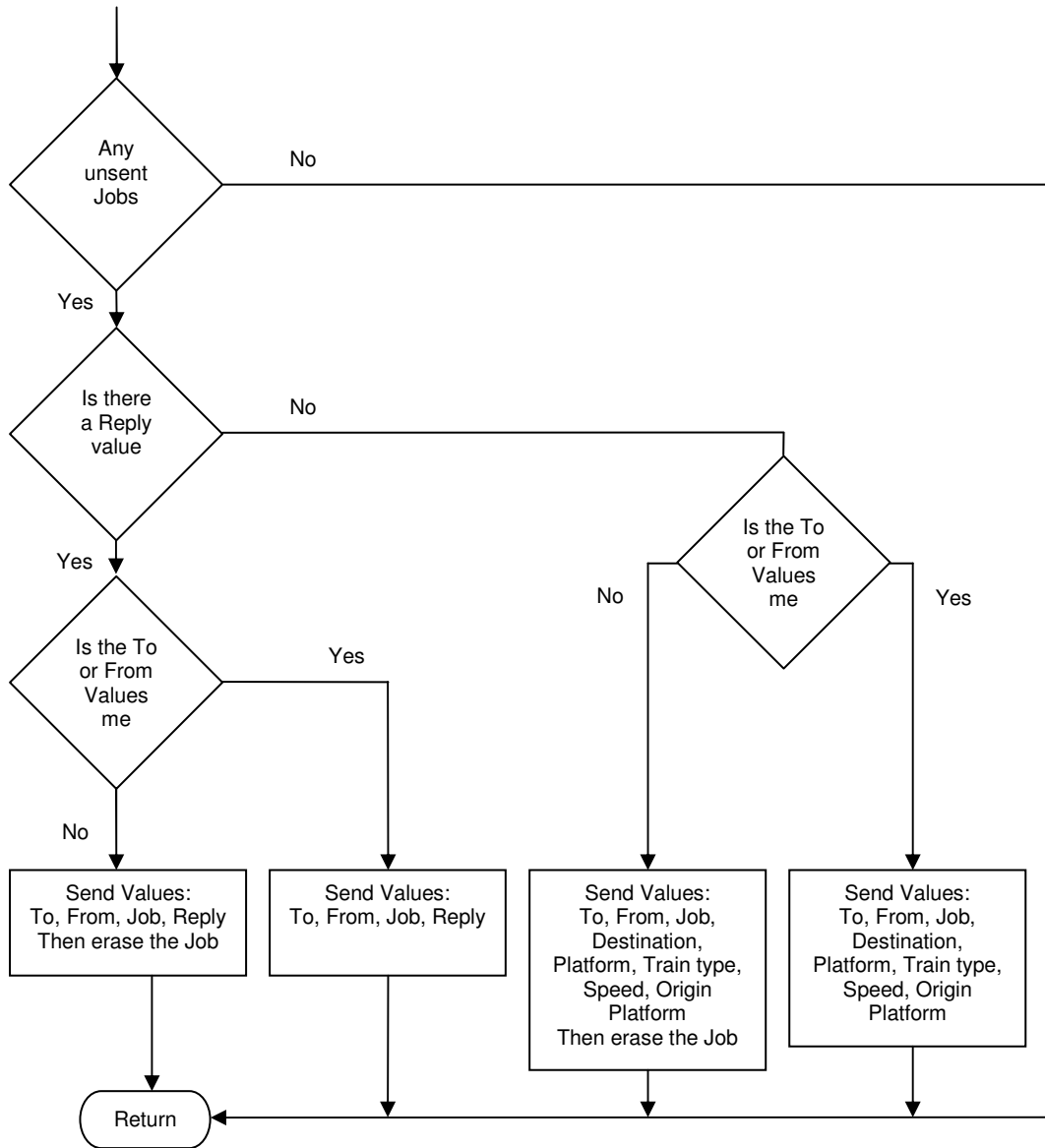
2.4. Movement Routines Flow Diagram



2.5. Receive Communications Flow Diagram

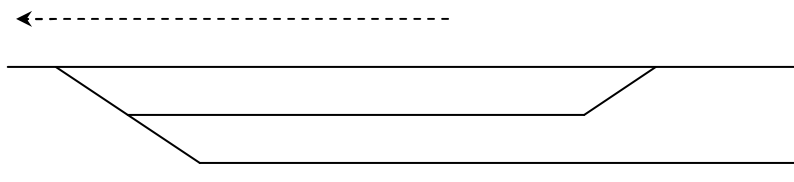


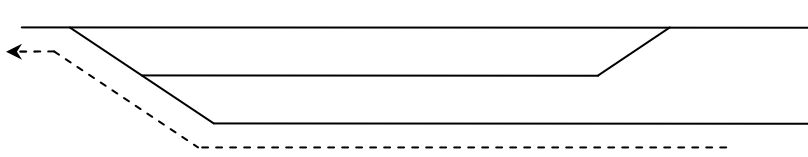
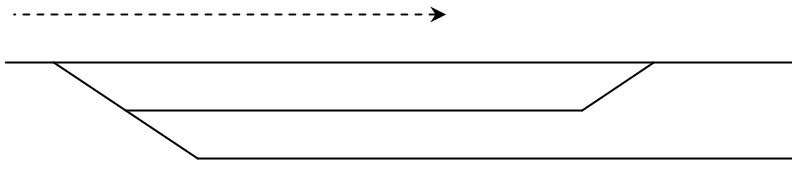
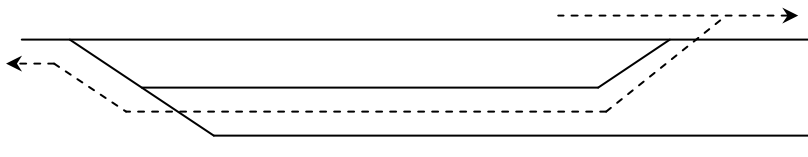
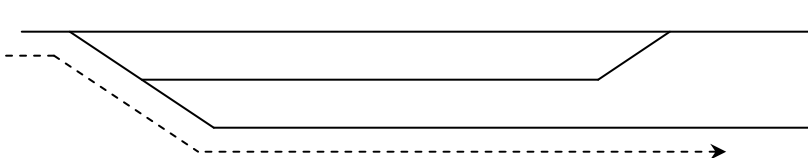
2.6. Send Communications Flow Diagram



2.7. Movement Routines

Routine A



Routine B**Routine C****Routine D****Routine E**

Appendix D: Test Results

During normal operation feedback is limited so it would be difficult to confirm whether a test had successfully failed or not. Additional debug messages have been included to prevent this limitation. The test system will record the response time to any user request to satisfy requirements R13, this will be recorded in the test comments.

Test Group 1 Description

Setting the train-type: A train is placed at the platform 'P' of the Block Controller 'BC'. A command is issued defined by the communication protocol to set the train-type. This satisfies requirements R11.

Results

The debug message confirms that the train type for the platform specified has been set.

Test	Variations	Successfully Failed	Comments
1	BC: 1 P: 1	No.	R13: 0.3 Pass.
2	BC: 1 P: 2	No.	R13: 0.3 Pass.
3	BC: 2 P: 1	No.	R13: 0.4 Pass. This has also tested the relay functionality of the communications.
4	BC: 2 P: 2	No.	R13: 0.3 Pass. See comments for test 3.

Test Group 2 Description

Setting the train-type: A train is NOT placed at the platform 'P' of the Block Controller 'BC'. A command is issued defined by the communication protocol to set the train-type. This satisfies requirements R11.

Results

The debug message confirms that the train type for the platform specified has NOT been set as no train is found.

Test	Variations	Successfully Failed	Comments
5	BC: 1 P: 1	No.	R13: 0.3 Pass.
6	BC: 1 P: 2	No.	R13: 0.3 Pass.
7	BC: 2 P: 1	No.	R13: 0.3 Pass. This has also tested the relay functionality of the communications.
8	BC: 2 P: 2	No.	R13: 0.4 Pass. See comments for test 7.

Test Group 3 Description

Setting the train's destination: The train at the platform 'P' of Block Controller 'BC' has had its train-type set. A command is issued defined by the communication protocol to set the destination of this train. This satisfies requirements R9.

Results

The debug message confirms that the destination of the train at the platform specified has been set.

Test	Variations	Successfully Failed	Comments
9	BC: 1 P: 1	No.	R13: 0.3 Pass.
10	BC: 1 P: 2	No.	R13: 0.4 Pass.
11	BC: 2 P: 1	No.	R13: 0.3 Pass. This has also tested the relay functionality of the communications.
12	BC: 2 P: 2	No.	R13: 0.3 Pass. See comments for test 11.

Test Group 4 Description

Setting the train's destination: The train at the platform 'P' of Block Controller 'BC' has NOT had its train-type set. A command is issued defined by the communication protocol to set the destination of this train. This satisfies requirements R9.

Results

The debug message confirms that the destination of the train at the platform specified has NOT been set as not train-type has been set.

Test	Variations	Successfully Failed	Comments
9	BC: 1 P: 1	No.	R13: 0.3 Pass.
10	BC: 1 P: 2	No.	R13: 0.4 Pass.
11	BC: 2 P: 1	No.	R13: 0.3 Pass. This has also tested the relay functionality of the communications.
12	BC: 2 P: 2	No.	R13: 0.3 Pass. See comments for test 11.

Test Group 5 Description

Departing a train: The train at the platform 'P' of Block Controller 'BC' has had its train-type and destination set. A command is issued defined by the communication protocol to depart this train. This satisfies requirements R10.

Results

The debug message confirms that the train at the platform specified will be departing.

Test	Variations	Successfully Failed	Comments
13	BC: 1 P: 1	No.	R13: 0.3 Pass.
14	BC: 1 P: 2	No.	R13: 0.3 Pass.
15	BC: 2 P: 1	No.	R13: 0.3 Pass. This has also tested the relay functionality of the communications.
16	BC: 2 P: 2	No.	R13: 0.3 Pass. See comments for test 11.

Test Group 6 Description

Departing a train: The train at the platform 'P' of Block Controller 'BC' has had either its train-type or destination NOT set. A command is issued defined by the communication protocol to depart this train. This satisfies requirements R10.

Results

The debug message confirms that the train at the platform specified will NOT be departing as one of the prerequisites was not supplied.

Test	Variations	Successfully Failed	Comments
17	BC: 1 P: 1	No.	R13: 0.3 Pass.
18	BC: 1 P: 2	No.	R13: 0.3 Pass.
19	BC: 2 P: 1	No.	R13: 0.3 Pass. This has also tested the relay functionality of the communications.
20	BC: 2 P: 2	No.	R13: 0.2 Pass. See comments for test 19.

Test Group 7 Description

Movement Routines: The train at the platform 'P' of Block Controller 'BC' has had its destination 'D' set which includes a destination platform 'DP'. The train-type has been set a double ended train and destination platform 'DP' is clear of any other trains. It has been issued with a depart command.

Results

In order.

The debug message confirms that the train at platform 'P' is being departed.

The debug message confirms a message is sent to Block Controller 'D'.

The debug message confirms the request is granted.

The debug message confirms the Job has been added to the Job list of 'D'.

Points will be set for a clear passageway at both 'BC' and 'D'. (if necessary)

The signal of platform 'P' will be set to Green.

The train from platform 'P' will begin to move

The signal of platform 'P' will be set to Red once the train has passed.

Points will close once the train has left Block Controller 'BC' (if necessary)

The debug message confirms the Job has been removed from the Job list of 'BC'.

The debug message confirms platform data for 'P' will be removed (update).

The debug message confirms platform data for 'DP' will be added to (update).

The train will stop at platform 'DP' of Block Controller 'D'

The debug message confirms the Job has been removed from the Job list of 'D'.

Test	Variations	Successfully Failed	Comments
21	BC: 1 P: 1 D: 2 DP: 1	No.	
22	BC: 1 P: 1 D: 2 DP: 2	No.	
23	BC: 1 P: 2 D: 2 DP: 1	No.	
24	BC: 1 P: 2 D: 2 DP: 2	No.	
25	BC: 2 P: 1 D: 1 DP: 1	No.	

26	BC: 2 P: 1 D: 1 DP: 2	No.	
27	BC: 2 P: 2 D: 1 DP: 1	No.	
28	BC: 2 P: 2 D: 1 DP: 2	No.	

Test Group 8 Description

Movement Routines: The train at the platform 1 of Block Controller 'BC' has had its destination Block Controller 'D' set which includes a destination platform of 1. The train-type has been set a single ended train and destination platform 1 is clear of any other trains. It has been issued with a depart command.

Results

In order.

The debug message confirms that the train at platform 'P' is being departed.

The debug message confirms a message is sent to Block Controller 'D'.

The debug message confirms the request is granted.

The debug message confirms the Job has been added to the Job list of 'D'.

The signal of platform 1 will be set to Green.

The train from platform 1 will begin to move

The signal of platform 1 will be set to Red once the train has passed.

The debug message confirms the Job has been removed from the Job list of 'BC'.

The debug message confirms platform 1's data of 'BC' will be removed (update).

The debug message confirms platform 1's data of 'D' will be added to (update).

The train will stop at platform 1 of 'D'

The debug message confirms a rap around is required.

A new job request is sent to 'BC'

The debug message confirms the request is granted.

The train will uncouple and move to Far-Home of 'D'.

All points will open at 'D'.

The train will begin to move out of Block Controller 'D' and will stop at platform 1 of 'BC'.

All points of 'D' will then close.

The debug message confirms platform 1's data of 'BC' will be added to (update).

A new job request is sent to Block Controller 'D' to send the train back.

The debug message confirms the request is granted.

The signal of platform 1 of 'BC' will be set to Green.

The train from platform 1 will begin to move.

The signal of platform 1 will be set to Red once the train has passed.

The debug message confirms the Job has been removed from the Job list of 'BC'.

The debug message confirms platform 1's data of Block Controller 'BC' will be removed (update).

The debug message confirms platform 1's data of Block Controller 'D' will be added to (update).

The train will couple onto the railway carriage at platform 1 of 'D'.

Test	Variations	Successfully Failed	Comments
29	BC: 1 D: 2	No.	Uncoupling is risky!
30	BC: 2 D: 1	No.	See comments for test 29.

Test Group 9 Description

Conflict Movement Routines: The train at the platform 'P' of Block Controller 'BC' has had its destination 'D' set which includes a destination platform 'DP'. The train-type has been set a double ended train and destination platform 'DP' is NOT clear of any other trains. It has been issued with a depart command.

Results

In order.

The debug message confirms that the train at platform 'P' is being departed.

The debug message confirms a message is sent to Block Controller 'D'.

This is only granted when the platform is freed...

The debug message confirms the Job has been added to the Job list of 'D'.

Points will be set for a clear passageway at both 'BC' and 'D'. (if necessary)

The signal of platform 'P' will be set to Green.

The train from platform 'P' will begin to move

The signal of platform 'P' will be set to Red once the train has passed.

Points will close once the train has left Block Controller 'BC' (if necessary)

The debug message confirms the Job has been removed from the Job list of 'BC'.

The debug message confirms platform data for 'P' will be removed (update).

The debug message confirms platform data for 'DP' will be added to (update).

The train will stop at platform 'DP' of Block Controller 'D'

The debug message confirms the Job has been removed from the Job list of 'D'.

Test	Variations	Successfully Failed	Comments
31	BC: 1 P: 1 D: 2 DP: 1	Yes.	R13: Infinite timed therefore Failed. There is a design limitation where the system will hang as there is another train blocking DP. As the test doesn't define moving the blocking train this test will never complete. See Post Mortem.
32	BC: 1 P: 1 D: 2 DP: 2	Yes.	See comments for test 31.
33	BC: 1 P: 2 D: 2 DP: 1	Yes.	See comments for test 31.
34	BC: 1 P: 2 D: 2 DP: 2	Yes.	See comments for test 31.

35	BC: 2 P: 1 D: 1 DP: 1	Yes.	See comments for test 31.
36	BC: 2 P: 1 D: 1 DP: 2	Yes.	See comments for test 31.
37	BC: 2 P: 2 D: 1 DP: 1	Yes.	See comments for test 31.
38	BC: 2 P: 2 D: 1 DP: 2	Yes.	See comments for test 31.

Appendix E: Original Project Proposal

Student Name: David Graham

Proposed Supervisor: Peter Knaggs

Course: Computing

Project: Investigation into software abstraction is computer hardware control.

Project Aims and Objectives:

Controlling a model train around a simple layout using a PIC Microcontroller and displaying its location on a Mimic displayed on a PC. Train location will be detected by reflective switches, LEDs will be used as signals and points will be driven by off the shelf points motors. (The hardware side of this project will be complete by the start of this project as it does not focus on it but instead focuses on programming (Software Dev)). Following needs to be discussed with a tutor: I intent to explore the benefits of abstraction, meaning while the PIC will initially do the full control and logic, the PC will only be listening to the PIC and displaying the location of the train. I will then remove this abstraction so the PC decides the logic while also displaying the mimic as before and the PIC simply acts as an interface to the hardware, I will relate it to real world situations where such abstraction is used as a Failsafe system, where failure of one system doesn't result in failure of the whole system. Somehow I will measure the difference.

Outside organisation(s) involved, if any, and main contact:

None

Special University resources (hardware and software) required:

PC

List of Deliverable (e.g. Requirements, Design, Specification, ect):

Software Life Cycle excluding support – two configurations for PIC/PC

Outline of methods:

Research embedded software and code optimisation

Research real world scenarios

Creating a Mimic on a PC platform

Serially communicating between PIC and PC

What is distinctive /honours worthy about the proposed project?

Embedded Software Control

Appendix F: Original Project Plan

Title

Does abstracting control reduce the complexity of a railway control system?

Introduction

This project focuses on control of model trains along an undefined route. Traditional automated train control has relied on a preset route and timings which can cause a headache to plan and any changes are time consuming to implement. The project describes how using abstraction in hardware the trains can be routed in a more intelligent way. The microcontroller based control architecture will be designed to be scalable for large and complex model railways layouts. The project is targeted at any model railway hobbyist who requires automating their railway layout.

I have no great interest in model railways but I am more interested in computer control. There are few things that can be easily controlled by a computer so a model railway seems the easiest to control. This solves a real problem with model train control and has great scope to be taken further when I graduate. The principal of routing a train has a number of links with computing principals which makes my job easier as theories and solutions have already been created which I can recreate. Control had a big part of my placement year and again I can use a lot of what I learnt on this project for example state machines in control routines.

Background

For a model railway hobbyist, creating a railway layout that looks and acts as close to the real thing as possible is their main aim. The expense of the hobby can put off children who you would expect it to be targeted at. The main audience for this hobby is generally the older generation who have the disposable income that is required. With the aid of lots of money creating a layout with realistic scenery is very achievable however control of the trains themselves is a more specialist area. Both Hornby and Peco, the market leaders in model railway products, don't offer any kind of automation control products apart from their power controllers which are manually controlled. The lack of control products has meant that the average layout uses an unrealistic circular design. This not only causes repetitive movements of the trains which can be dull to watch but the layouts themselves can take up most of the room

that they are housed in. Without any sort of automation it is currently not possible for example to have a layout run down only one side of a room. This type of layout would be far more convenient for most people as it doesn't mean reserving a whole room for the layout.

The project is focused around the control of an OO gauge model railway. OO has a scale ratio of 1:176.2 and runs on a 12V DC supply. There are two rails which depending which one is live, controls the direction of the trains travel.

Project Objectives

A modularised control system will be created to control a model railway.

Routing of the trains should be done intelligently with no interaction from an operator.

Ideally it should be easy to add a new block if the layout expands and the complete system should be made aware of the change in a short time.

Priority 1: Movement of a train from one block to another. Scheduling is done via a Personal Computer with a basic Set-and-Depart interface. No advance scheduling can be achieved.

Priority 2: The PC's interface should be more advanced with scheduling being achieved along time in advance of the train departing.

Priority 3: Changes to the system, for example addition of new blocks and destinations can be downloaded from the PC to the block-controllers on initialisation.

Constraints

The trains run on a 12V DC track.

The signals use 1.2V LEDs.

The point motors operate using a 'burst' of a 12V DC current. If the current is left on the motors will overheat.

The system has to be safe.

Method

The problem that I am trying to solve has already been solved on the full scale rail network. We don't have a central control centre that controls the whole of the UK's

railways. Instead we have much smaller control centres traditionally called signal boxes. These control only a small section of track that can be easily managed by an signal man or control system. I am therefore going to take the same solution by breaking any model railway layout into blocks. Each block will be controlled by a microcontroller based block-controller that I will also build. All the block-controllers will be able to communicate with each other via a serial connection. Communication is required in the event that a train has to cross the boundary of its associated block-controller. With full scale railways the signal box doesn't actually control the train as the train driver does, however I intend the block-controller to also act as the train driver by setting the speed and power of the train.

Over the summer I have created a demo layout consisting of 2 blocks. The blocks are mirrors of each other and are made up of a basic layout with 3 points, 5 sensors and 2 signals.

Deliverables

Requirements Specification

User Guide

Design Documentation

High Level System

Block-Controller

PC Scheduler Interface

Products

Block-Controller Hardware

PIC18F2550 Firmware (Firmware / C)

Windows Scheduling Software (Windows / C#)

Activities

See Individual Project Plan Timeline

Resources

MPLAB IDE: This is Microchip's own Integrated Development Environment for developing firmware for any PIC microcontroller.

MPLAB C18 compiler: This is the compiler used to compile C code into a hex format that can be loaded onto a PIC microcontroller. The student edition is free, however after 60 days the advance features for optimising performance are disabled.

Visual Studio: This is used to create and compile the Windows scheduler application that will run on the desktop computer. This is available free for students from MSDNAA.

Risk Analysis

Risk	Risk Probability	Risk Impact	Risk Factor
Unavailability of resources	2	8	16
Unavailability of key personnel	4	5	20
Technical problems	6	6	36
Other tasks demanding my time	6	5	30

Unavailability of resources: All the resources I need for this project I currently have and it is very unlikely that these will be taken away from me and as such I have no contingency measures in place if this would occur.

Unavailability of key personnel: The main issue in this area is if my supervisor or reader would be made to resign. In the event I would be allocated another lecturer to fill the gap but my project is likely to suffer as a result.

Technical problems: This is very likely as I am not only dealing with software but also hardware. When something smokes an all-nighter will not fix the problem. As with all fixable problems I intend to work at it until I have solved the problem. This will delay my progress but I have to manage my progress and alter my plan to guide me back onto track. To try to prevent this I have done lots of research in the areas that I intend to use which should help me when I come around to using it.

Other tasks demanding my time: This is a little unknown as I am not yet at a stage where I can plan for this as not all the assignments have been issued. When I can, to prevent too much time being taken away from this project I intend to plan my time wisely and any changes will be consulted with the plan and it revised.

Appendix G: Predicted Project Timeline

Week	Date	Activity	Prerequisite	Est.time / Weeks
1	08/10/07	Supervisor Sign off	Project Proposal	
2	15/10/07			
3	22/10/07	Reader and Plan Sign off	Supervisor Sign off	
4	29/10/07	Comms Spec Complete	Project Plan	2
5	05/11/07			
6	12/11/07	Firmware Spec Complete	Project Plan	1
7	19/11/07	Firmware Coding	Specifications	5
8	26/11/07			
9	03/12/07	Coding Progress review	Coding	
10	10/12/07			
11	17/12/07			
12	24/12/07	Holidays		
13	31/12/07	Holidays		
14	07/01/08	Testing & Debug	Firmware complete	2
15	14/01/08			
16	21/01/08	First draft write up	Firmware & testing complete	5
17	28/01/08			
18	04/02/08			
19	11/02/08			
20	18/02/08			
21	25/02/08	Write up tidy up	First draft write up complete	2
22	03/03/08			
23	10/03/08	Write up complete	Write up complete	1
24	17/03/08	Binding	Binding complete	1
25	24/03/08	Hand in end of week	Project complete	1
26	31/03/08	Creation of poster		1
27	07/04/08			
28	14/04/08			
29	21/04/08			
30	28/04/08			
31	05/05/08	Defence/Demo 5 th – 9th		

Appendix H: Actual Project Timeline

Week	Date	Activity	Notes
1	08/10/07	Supervisor Sign off	
2	15/10/07		
3	22/10/07	Reader and Plan Sign off	
4	29/10/07		
5	05/11/07	Comms Protocol Completed	
6	12/11/07		Project development suspended
7	19/11/07	Assignment Due	DM assignment hand in
8	26/11/07	Movements Routine defined	
9	03/12/07		Project development suspended
10	10/12/07	Assignment Due	NS assignment hand in
11	17/12/07	Block Controller Coding	
12	24/12/07		Christmas
13	31/12/07	Block Controller Coding	
14	07/01/08	Coding Review	
15	14/01/08		Project development suspended
16	21/01/08	Assignment Due	IAD assignment hand in
17	28/01/08	Block Controller Coding	
18	04/02/08	Write up begins	Notes have been taken up until now
19	11/02/08	Block Controller Coding	
20	18/02/08	Coding Review	
21	25/02/08		Project development suspended
22	03/03/08	Assignment Due	CGT assignment hand in
23	10/03/08	Write up	
24	17/03/08	Write up	
25	24/03/08	Binding & Hand in	
26	31/03/08	Creation of poster	
27	07/04/08		
28	14/04/08		
29	21/04/08		
30	28/04/08		
31	05/05/08	Defence/Demo 5 th – 9th	

Appendix I: Project Diary

Friday 28th March 2008

Handed in. Hurray!

Tuesday 25th March 2008

Binding commencing today.

Tuesday 4th March 2008

Finalising the write up begins today. T Minus 4 weeks.

Tuesday 5th February 2008

The last few days I have been concentrating on coding the movement routines and also fixing some bugs with the 'sending' comms. I have made good progress on both.

Saturday 2nd February 2008

Yesterday and today I have been working on the Platform manager. I can now set the details of a train at a platform and request the controller to depart the train. There is a small problem with one of the messages but overall it seems to be coming together nicely.

Wednesday 30th January 2008

I spent most of today updating the documentation with the new terminology.

Tuesday 29th January 2008

Finally getting some movement routines completed without train movements, which means points/signals/sensors all operate depending on the requested platform. Comms is starting to look good now. Had a major problem with Comms forwarding -- This is where the controller forwards the whole message it receives to the next controller when the TO element of the xml isn't the ID of the current controller. Originally I copied the message and simply sent it back to the next controller, But there wasn't the memory needed to copy a 200 char string twice. The reason I had to do a copy instead of sending the original was the string tok function has a side effort of breaking the string so it can't be reused. To over come this the message is loaded into the Jobs queue as a normal message would be, but instead of doing anything with it the send comms functions spits in back out, problem solved.

Sunday 27th January 2008

Been working a lot on the Comms for the last three days. Starting to come against memory problems and I am starting to realise where I can cut down on wasted memory.

One thing I am starting to remove is getter and setting functions as these take up wasted memory. I have now changed some of the terminology of the project, wishes are now jobs however there is a lot of code and documentation that still features wishes. The Capacitor Discharge Units have arrived which should solve a problem I had with the points taking too much current.

Tuesday 25th December 2007

Merry Christmas. To get some time to my own I have been coding parts of the Jobs List.

Monday 5th November 2007

As both controllers are sharing one power supply, and as when a point is powered it draws the full current of the supply, if both controllers move a point at the same time like on initialisation, there will not be enough current to power both points and therefore one will fail to move.

Tuesday 30th October 2007

The project plan was approved today. I am now looking for a reader.

Friday 20th October 2007

I have started to create a project plan.

Monday 15th October 2007

Continuing defining the communication protocol.

Thursday 11th October 2007

The project has now been approved by my university lecture.

Tuesday 11th September 2007

I am now able to uncouple and reattach a railway carriage.

Monday 10th September 2007

Was able to get a basic forward/backward movement working with the new demo layout. It can be seen here. Also got the timers working correctly.

Sunday 9th September 2007

Finished the second demo track.

Friday 7th September 2007

Solved the problem with the comms which was due to a common ground not being used between demo board and controller which explains why it worked on the demo board but not on the controller. Started work on finishing the second demo track.

Thursday 30th August 2007

Still trying to debug the problem with the comms with no success.

Wednesday 29th August 2007

I created the first Block controller today. Initial tests seem fine with most I/O, but I have a strange problem with the comms where it works fine on the demo board but when I move the PIC over to the new controller it doesn't work.

Monday 27th August 2007

I have achieved PIC to PIC comms. I can now type a key on the keyboard which sends it to PIC 1, that relays it to PIC 2 which relays it back to the computer to be displayed on the screen. This proves that a ring architecture is achievable.

Sunday 26th August 2007

I have been refining the PC to PIC comms, I have now started work on PIC 2 PIC comms.

Thursday 23rd August 2007

I soldered the serial driver to my demo board and was able to output text to the windows hyper terminal.

Tuesday 21st August 2007

I'm happy with timers and interrupts now so I have moved onto testing the input lines for the sensors.

Wednesday 15th August 2007

Now I'm working with timers and interrupts

Monday 13th August 2007

Got both PWMs working today, now I am working on using timers.

Thursday 9th August 2007

Continued creating a demo board, had some issues with Master Clear Line being enabled as this created flickering of the LEDs which I debugged in the late afternoon. Also had trouble with getting I/O A6 to work, this is multiplexed with the oscillator, setting the Oscillator configuration bit to "INTOSC: INTOSC+RA6, USB-EC" solved this.

Wednesday 8th August 2007

Started to create a small demo board to test the PIC18.

Tuesday 7th August 2007

Started to create a program structure to prove I can use all the ports I want to use.

Monday 6th August 2007

Painted the base board today.

Sunday 5th August 2007

The last couple days I have been working on strip board and chip diagrams.

Friday 28th July 2007

Development has been slow recently. Time has been taken up understanding the new PIC18 microcontroller. Initially I have general issues with programming the new chip, which I solved when I found out I needed to turn off the low voltage programming feature. I then tried to flash a LED, which after a lot of time wasted I found that the chip was running so fast that this was hard to do; so I lowered the speed of the chip. I have also had issues with exporting the config bits from MPLAB IDE, which I have now solved. I have now been able to demo a Signal.

Thursday 19th July 2007

I have decided to go with a modularised design by using two microcontrollers, controlling their own section of track. I can talk about control abstraction and whether it aids the control process.

Wednesday 11th July 2007

No progress today, but the Darlington driver for the points control did fail and had to be replaced. To make sure this doesn't happen again two Darlington drivers will be placed in parallel to decrease the load on them.

Tuesday 10th July 2007

Today I worked on getting the points working. They were already wired in but just needed a software update.

Monday 9th July 2007

More Opto Switches arrived in the post today so I was able to construct a forward/backward configuration but using two Opto switches.

Saturday 7th July 2007

This weekend a prototype of the hardware was constructed using a smaller PIC Microcontroller. One Opto switch was used and basic forward/backwards train control was achieved. The speed of the train was initially slow however and it was decided to replace the power supply with one that delivered 1200mA instead of 500. The H Bridge was running extremely hot throughout testing and eventually unsoldered itself. Reducing the heat will have to be considered in future builds.

Appendix J: CD Contents

Dissertation Directory

This folder contains a portable document format file of this dissertation.

Dissertation.pdf

Source Directory

This folder contains the C source code.

Comms.c	CommsReceive.c	CommsSend.c	Identification.c
JobManager.c	Main.c	MovementRoutineA.c	MovementRoutineB.c
MovementRoutineC.c		MovementRoutineD.c	MovementRoutineE.c
Movements.c	PlatformManger.c	Points.c	Sensors.c
Setup.c	Signals.c	Timer.c	TrainControl.c

Source/H Directory

This folder contains the header files.

Comms.h	CommsReceive.h	CommsSend.h	Identification.h
JobManager.h	MovementRoutineA.h	MovementRoutineB.h	MovementRoutineC.h
MovementRoutineD.h		MovementRoutineE.h	Movements.h
MsgTypes.h	PlatformManger.h	Points.h	Sensors.h
Setup.h	Signals.h	Timer.h	TrainControl.h
Types.h			

Deliverables Directory

This folder contains the firmware for both Block Controllers in hex format.

Weymouth.hex

London.hex

Media Directory

This folder contains a few images and a video of the system in motion.